XXI JORNADAS DE INGENIERÍA DEL SOFTWARE Y BASES DE DATOS

Jesús J. García Molina (Ed.)

JISBD

AQUILAFUENTE

Ediciones Universidad Salamanca

XXI Jornadas de Ingeniería del Software y Bases de Datos

JESÚS J. GARCÍA MOLINA (ED.)

XXI Jornadas de Ingeniería del Software y Bases de Datos

Ediciones Universidad Salamanca

AQUILAFUENTE, 219

Motivo de cubierta:
Diseñadora María Alonso Miguel

1.º edición: septiembre, 2016

ISBN: 978-84-9012-627-1 (PDF)

Ediciones Universidad de Salamanca
www.eusal.es
eusal@usal.es

Realizado en España – Made in Spain

Obra sometida a proceso de
evaluación mediante sistema de revisión por pares a ciegas a tenor de las normas del congreso

♠

Ediciones Universidad de Salamanca es miembro de la UNE
Unión de Editoriales Españolas www.une.es

♠

Catalogación de editor en ONIX accesible en https://www.dilve.es/

A José María Troya

En una comida en Sevilla con José María Troya, Pere Botella e Isidro Ramos y en el marco de unas jornadas de trabajo organizadas por Miguel Toro decidimos iniciar formalmente la presentación de trabajos de investigación en Ingeniería del Software en forma de un congreso anual: las Jornadas de Ingeniería del Software, las que posteriormente, con la fusión de las Jornadas de Bases de Datos, darían origen a las JISBD y posteriormente a SISTEDES.

Algo se hizo bien y no fue fruto de la casualidad. Existían las bases suficientes para emprender nuestro particular viaje a Ithaca.

José María, desde su época de estudiante de la Especialidad de Cálculo Automático de la UCM, y más tarde durante la realización de su tesis doctoral, en la que resolvió un problema difícil definiendo una regla heurística para rebajar su complejidad computacional, se mostró como un trabajador infatigable, con una capacidad de iniciativa propia, una ética profesional a toda prueba y una sonriente amabilidad.

Los frutos de su posicionamiento personal y científico, excelentes y públicos, se acompañaron siempre con una amabilidad y seriedad que fraguaron el clima de amistad que caracteriza a nuestra comunidad. Gracias también por eso, José María.

Nosotros no hemos llegado todavía a Ithaca. Tú te anticipaste también en esto. Nos atrevemos a decir, recordando nuestras experiencias latinoamericanas juntos, que "te fuiste pronto como los elegidos en plena gloria y juventud"

Los versos de Kavafis en los que usa el Viaje como metáfora de la Vida no son capaces de llenar el vacío que nos has dejado, pero captan tu particular singladura:

"Cuando emprendas tu viaje a Ithaca /pide que el camino sea largo, /lleno de aventuras, lleno de experiencias/No temas a los lestrigones ni a los cíclopes / ni al colérico Poseidón, /seres tales jamás hallarás en tu camino, /si tu pensar es elevado, si selecta /es la emoción que toca tu espíritu y tu cuerpo/Ten siempre a Ithaca en tu mente. /Llegar allí es tu destino."

Pero, con celeridad y silencio nos has dejado:

"Mas no apresures nunca el viaje/Mejor que dure muchos años /y atracar, viejo ya, en la isla, enriquecido de cuanto ganaste en el camino /sin esperar que Ithaca te enriquezca"

Tu ausencia estará siempre presente en JISBD, la cuales ayudaste a crear.

Descansa en Paz

Pere Botella, Isidro Ramos y Miguel Toro

Prólogo

Las "Jornadas de Ingeniería del Software y Bases de Datos" (JISBD) constituyen el foro que cada año reúne a la comunidad científica española en las áreas de Ingeniería del Software y Bases de Datos y siempre han atraído el interés de grupos de investigación de Portugal e Iberoamérica en estas dos áreas. JISBD es organizada por la Sociedad de Ingeniería de Software y Tecnologías de Desarrollo de Software (SISTEDES) junto a otras dos conferencias: Jornadas de Programación Declarativa (PROLE) y Jornadas de Ciencia e Ingeniería de Servicios (JCIS). La vigésimo primera edición de JISBD es uno de los quince eventos científicos que integran la IV Conferencia Española de Informática (CEDI) que se celebra en Salamanca. CEDI tiene una periodicidad de tres años y su propósito es mostrar a la sociedad el estado actual de la informática en España.

En la edición actual, JISBD ha continuado con la organización basada en áreas temáticas o "tracks" puesta en marcha en la edición previa. Los tipos de contribución han sido los considerados en ediciones anteriores: artículos completos, artículos cortos, artículos relevantes y demos. Como novedad se realizó un llamamiento a nuevos tracks dentro de la primera solicitud de contribuciones lanzada a principios de noviembre de 2015. Dado que no se recibió ninguna solicitud no fue necesario aplicar el mecanismo previsto para dar cabida a nuevos tracks. Por tanto, JISBD'2016 incluye los mismos tracks que en la edición anterior: *Arquitecturas del Software y Variabilidad*, *Calidad y Pruebas*, *Desarrollo de Software Dirigido por Modelos*, *Gestión de Datos*, *Ingeniería del Software Guiada por Búsqueda*, *Ingeniería Web y Sistemas Pervasivos* y *Procesos Software y Metodologías*; y de nuevo se ha incluido el track *Abierto* para dar cabida a los trabajos que no encajan en ninguno de los tracks anteriores.

Otra novedad ha tenido que ver con la modalidad de "artículos relevantes" ya publicados en revistas con índices de impacto o conferencias internacionales prestigiosas en las áreas asociadas a un determinado track. Con el fin de facilitar el proceso de selección y asegurar la calidad de estas contribuciones, se ha establecido que un "trabajo relevante" debe haber sido publicado en una revista en el cuartil Q1 de JCR o en una de las dos conferencias que ha seleccionado cada track.

Cabe destacar un incremento significativo en el número de contribuciones recibidas con respecto a las tres ediciones anteriores. Mientras en 2015 se recibieron 70 contribuciones, 54 en 2014 y 64 en 2013, en esta edición se han recibido 94 contribuciones (34 completos, 29 cortos, 25 relevantes y 6 demos). El número de trabajos aceptados ha sido 79 (30 completos, 21 cortos, 24 relevantes y 5 demos). Estos números parecen avalar la nueva organización en torno a áreas temáticas y que las JISBD pueden jugar un importante papel para dinamizar las diferentes comunidades relacionadas con la ingeniería del software y las bases de datos en nuestro país.

La conferencia invitada será impartida por Andrei Voronkok, prestigioso investigador de la Universidad de Manchester que ha recibido el premio Herbrand por sus contribuciones al razonamiento automático y que es creador de EasyChair, una de las herramientas de gestión de conferencias más extendidas en el mundo. El Dr. Voronkok analizará los desafíos a los que se ha debido hacer frente en la construcción de EasyChair desde el punto de vista del diseño de software y la gestión de los datos, así como de los retos para el futuro. Además, las conferencias invitadas de JCIS (Tommi Mikkonnen, Institute of Pervasive Computing, Tampere, Finland) y de PROLE (Arnaud Gotlieb, Simula Research Laboratory, Norway), que se celebran en paralelo en esta ocasión, son también parte del programa de JISBD'2016.

Dada la estructura actual de JISBD basada en track, toda la labor de organización es realizada por un equipo formado por el Presidente del Comité de Programa y los coordinadores de cada track (el listado aparece a continuación de esta presentación). Expreso mi agradecimiento a cada uno de los coordinadores de tracks por el esfuerzo que han realizado y por su buena disposición, ha sido un placer coordinar este equipo. El trabajo que he debido realizar ha requerido un contacto permanente con SISTEDES y con el Comité Organizador de CEDI'2016. Por un lado, debo agradecer a Fernando de la Prieta toda la ayuda prestada, como persona de contacto con dicho comité, para resolver todas las cuestiones relacionadas con la web, el uso de EasyChair, la edición de actas y la gestión económica. Por otro lado, contar con Oscar Díaz como enlace con SISTEDES me ha dado una gran tranquilidad en la toma de decisiones. Agradezco también el apoyo recibido en todo momento por Diego Sevilla.

Los agradecimientos finales para aquellos que son los principales protagonistas: autores y revisores. A los primeros por apoyar a JISBD con el envío de publicaciones y la presentación de sus trabajos en Salamanca, y a los segundos por su dedicación a la tarea de mantener el nivel de calidad esperado de las contribuciones a JISBD y ayudar a los autores a mejorar sus trabajos.

Y por último, esta presentación de JISBD'2016 no puede acabar sin recordar a José María Troya, uno de los impulsores de estas Jornadas y de la Ingeniería del Software en nuestro país.

Salamanca, 13 de septiembre de 2016

Jesús J. García Molina
Presidente del Comité de Programa

Comité de programa

Presidente

Jesús Joaquín García Molina (Universidad de Murcia)

Coordinadores de los tracks

Arquitecturas Software y Variabilidad: David Benavides (Universidad De Sevilla) y Jennifer Pérez Benedí (Universidad Politécnica de Madrid)

Calidad y Pruebas: Carme Quer (Universitat Politècnica de Catalunya) y María José Suárez-Cabal (Universidad de Oviedo)

Desarrollo de Software Dirigido por Modelos: Cristina Vicente Chicote (Universidad de Extremadura) y Juan de Lara (Universidad Autónoma de Madrid)

Gestión de Datos: Sergio Ilarri (Universidad de Zaragoza) y José Ramón Paramá (Universidad de A Coruña).

Ingeniería del Software Guiada por Búsqueda: José Raúl Romero Salguero (Universidad de Córdoba) y José Francisco Chicano García (Universidad de Málaga)

Ingeniería Web y Sistemas Pervasivos: Elena Navarro (Universidad de Castilla-La Mancha) y Roberto Rodríguez Echeverría (Universidad de Extremadura)

Proceso Software y Metodologías: Mercedes Ruiz (Universidad de Cádiz) y Agustín Yagüe (Universidad de Politécnica de Madrid)

Tema Abierto: Jesús J. García Molina (Universidad de Murcia)

Coordinador de demostraciones

Diego Sevilla Ruiz (Universidad de Murcia)

Enlace con SISTEDES

Óscar Díaz García (Universidad del País Vasco)

Comités de Revisión

Arquitecturas de Software y Variabilidad

Silvia Abrahao (Universitat Politécnica de Valencia)
David Benavides (Universidad de Sevilla)
Carlos Canal (Universidad de Málaga)
Rafael Capilla (Universidad Rey Juan Carlos)
Carlos E. Cuesta (Universidad Rey Juan Carlos)
Jessica Díaz (Universidad Politécnica de Madrid)
Lidia Fuentes (Universidad de Málaga)
José A. Galindo (INRIA)
Nadia Gamez (Universidad de Málaga)
Juan Garbajosa (Universidad Politécnica de Madrid)
Ruben Heradio (UNED, Universidad Nacional de Educación a Distancia)
Emilio Insfran (Universitat Politecnica de Valencia)
Juan Manuel Murillo (Universidad de Extremadura)
Elena Navarro (Universidad de Castilla-La Mancha)
Vicente Pelechano (Universidad Politécnica de Valencia)
Jennifer Pérez (Universidad Politécnica de Madrid)
Goiuria Sagardui (Universidad de Mondragon)
Pablo Trinidad (Universidad de Sevilla)
Salvador Trujillo (IKERLAN Research Centre)

Calidad y Pruebas

David Benavides (Universidad de Sevilla)
Raquel Blanco (Universidad de Oviedo)
Rubén Casado (Universidad de Oviedo)
M.J. Escalona (Universidad de Sevilla)
Jose L. Fernández Alemán (Universidad de Murcia)
Juan Garbajosa (Universidad Politécnica de Madrid)
Felix Garcia (Universidad de Castilla-La Mancha)
Marcela Genero (Universidad de Castilla-La Mancha)
Jose Ignacio Panach Navarrete (Universitat de València)
Oscar Pastor Lopez (Universitat Politécnica de Valencia)
Juan Sanchez (Universidad Politécnica de Valencia)
Sergio Segura (Universidad de Sevilla)
Ambrosio Toval (Universidad de Murcia)
Javier Tuya (Universidad de Oviedo)
Tanja E. J. Vos (Universidad Politécnica de Valencia)

Desarrollo de Software Dirigido por Modelos

Silvia Abrahão (Universidad Politécnica de Valencia)
David Ameller (Universidad Politécnica de Cataluña)
Orlando Ávila García (Tuguu)
Artur Boronat (Universidad de Leicester)
Javier Luis Cánovas Izquierdo (Universidad Oberta de Catalunya)
Pedro José Clemente Martín (Universidad de Extremadura)
Juan Garbajosa (Universidad Politécnica de Madrid)
Félix García(Universidad de Castilla la Mancha)
Antonio García-Domínguez (University of York)
Irene Garrigós Ferrández (Universidad de Alicante)
Gonzalo Génova Fuster (Universidad Carlos III)
Abel Gómez Llana (Universidad de Zaragoza)
Luis Iribarne Martínez (Universidad de Almería)
Miguel A. Laguna Serrano (Universidad de Valladolid)
Miguel A. de Miguel (Universidad Politécnica de Madrid)
Guadalupe Ortiz Bellot (Universidad de Cádiz)

Vicente Pelechano Ferragud (Universidad Politécnica de Valencia)
Antonia Mª Reina Quintero (Universidad de Sevilla)
José Raúl Romero (Universidad de Córdoba)
Goiuria Sagardui Mendieta (Universidad de Mondragón)
Jesús Sánchez Cuadrado (Universidad Autónoma de Madrid)
Ernest Teniente López (Universidad Politécnica de Cataluña)
Ambrosio Toval Álvarez (Universidad de Murcia)
Javier Troya (Universidad de Sevilla)
Salvador Trujillo (IKERLAN)
Antonio Vallecillo Moreno (Universidad de Málaga)
Juan Manuel Vara Mesa (Universidad Rey Juan Carlos)

Gestión de Datos

José Aldana Monters (Universidad de Málaga)
Antonio Corral (Universidad de Almería)
Francesco Guerra (Universidad de Módena y Reggio Emilia)
Arantza Illarramendi (Universidad del País Vasco)
Mercedes Martínez (Universidad de Valladolid)
José Norberto Mazón (Universidad de Alicante)
Philippe Roose (LIUPPA)
Ernest Teniente (Universitat Politècnica de Catalunya)
Raquel Trillo (Universidad de Zaragoza)
Belén Vela Sánchez (Universidad Rey Juan Carlos)
Yannis Velegrakis (Universidad de Trento)
José Ramón Ríos Viqueira (Universidad de Santiago de Compostela)
Marta Zorrilla (Universidad de Cantabria)

Ingeniería de Software Guiada por Búsqueda

Enrique Alba (Universidad de Málaga)
Isabel del Águila (Universidad de Almería)
José del Sagrado (Universidad de Almería)
Carmelo del Valle (Universidad de Sevilla)
José Javier Dolado (Universidad del País Vasco)
Antonia Estero-Botaro (Universidad de Cádiz)
Javier Ferrer (Universidad de Málaga)
Inmaculada Medina (Universidad de Cádiz)
Francisco Palomo (Universidad de Cádiz)
José Antonio Parejo (Universidad de Sevilla)
Daniel Rodríguez (Universidad de Alcalá)
Sergio Segura (Universidad de Sevilla)
Javier Tuya (Universidad de Oviedo)
Antonio Vallecillo (Universidad de Málaga)
Sebastián Ventura (Universidad de Córdoba)
Tanja Vos (Universidad Politécnica de Valencia)

Ingeniería Web y Sistemas Pervasivos

Rafael Berlanga (Universitat Jaume I)
Pedro J. Clemente (Universidad de Extremadura)
Rafael Corchuelo (Universidad de Sevilla)
María Valeria De Castro (Universidad Rey Juan Carlos)
M.J. Escalona (Universidad de Sevilla)
Jose Luis Garrido (Universidad de Granada)
Irene Garrigos (Universidad de Alicante)
Pascual González (Universidad de Castilla-La Mancha)
Francisco Luis Gutiérrez Vela (Universidad de Granada)
Luis Iribarne (Universidad de Almería)
Jon Iturrioz (Universidad del País Vasco)
Javier Jaen (Universidad Politécnica de Valencia)
Jose-Norberto Mazon (Universidad de Alicante)

Santiago Melia (Universidad de Alicante)
Ana Isabel Molina Díaz (Universidad de Castilla-La Mancha)
Patricia Paderewski (Universidad de Granada)
Juan Carlos Preciado (Universidad de Extremadura)
Fernando Sánchez (Universidad de Extremadura)
Ismael Sanz (Universitat Jaume I)

Proceso Software y Metodologías

Javier Dolado (Universidad del País Vasco)
M.J. Escalona (Universidad de Sevilla)
Juan Garbajosa (Universidad Politécnica de Madrid)
Félix García (Universidad de Castilla-La Mancha)
Patricio Letelier (Universidad Politécnica de Valencia)
Antònia Mas (Universidad de las Islas Baleares)
Ana Moreno (Universidad Politécnica de Madrid)
Elena Orta (Universidad de Cádiz)
Mario Piattini (Universidad de Castilla-La Mancha)
Francisco Ruiz (Universidad de Castilla-La Mancha)
María Isabel Sánchez (Universidad Carlos III)
Xavi Albaladejo (AXA-MEDLA)
Jorge Uriarte (Gailen)

Tema Abierto

Rafael Berlanga (Universitat Jaume I)
Nieves Brisaboa (Universidade de A Coruña)
José H. Canós (Universitat Politècnica de València)
Javier Luis Canovas Izquierdo (IN3 – UOC)
Yania Crespo (Universidad de Valladolid)
Óscar Dieste (Universidad Politécnica de Madrid)
María José Escalona (Universidad de Sevilla)
Lidia Fuentes (Universidad de Málaga)
Juan Garbajosa (Universidad Politécnica de Madrid)
Félix García (Universidad de Castilla-La Mancha)
Alfredo Goñi (Universidad del País Vasco)
Juan Hernández (Universidad de Extremadura)
Luis Iribarne (Universidad de Almería)
Santiago Melia (Universidad de Alicante)
Ana Moreno (Universidad Politécnica de Madrid)
Juan José Moreno (Universidad Politécnica de Madrid)
Vicente Pelechano (Universitat Politècnica de València)
Gustavo Rossi (LIFIA-F. Informatica. UNLP)
Goiuria Sagardui (Universidad de Mondragón)
José Samos (Universidad de Granada)
Pedro Sánchez (Universidad Politécnica de Cartagena)
Diego Sevilla (Universidad de Murcia)
Miguel Toro (Universidad de Sevilla)
Ambrosio Toval (Universidad de Murcia)
Juan Carlos Trujillo (Universidad de Alicante)
Javier Tuya (Universidad de Oviedo)
Ernest Teniente (Universitat Politècnica de Catalunya)

Índice

Arquitecturas Software y Variabilidad:

Calidad y Pruebas:

Desarrollo de Software Dirigido por Modelos:

Gestión de Datos:

Ingeniería del Software Guiada por Búsqueda:

Ingeniería Web y Sistemas Pervasivos:

Proceso Software y Metodologías:

Tema Abierto:

Arquitecturas Software y Variabilidad

Arquitecturas Software y Variabilidad
Coordinadores: David Benavides y Jennifer Pérez Benedí

Nieves R. Brisaboa, Alejandro Cortiñas, Miguel R. Luaces y Óscar Pedreira.
Aplicando Scaffolding en el desarrollo de Líneas de Producto Software. (Completo)

Héctor Humanes, Iván Hernández, Jessica Díaz, Jennifer Perez, Alfonso Ríos, Javier Gonzalez-Rodriguez y Jordi Paraire. *Estudio del Soporte a la Variabilidad en la Nube en un entorno Multitenencia: Plataforma GPaaS*. (Completo)

José Miguel Horcas, Mónica Pinto y Lidia Fuentes.
Evolución arquitectónica de servicios basada en modelos CVL con cardinalidad. (Completo)

Jorge L. Rodas, Javier Olivares, José A. Galindo y David Benavides.
Hacia el uso de sistemas de recomendación en sistemas de alta variabilidad. (Corto)

Javier Criado, Silverio Martínez-Fernández, David Ameller y Luis Iribarne.
Measuring the quality of transformation alternatives in software architectures evolution. (Corto)

Mariuxi Vinueza, Jorge L. Rodas, José A. Galindo y David Benavides.
El uso de modelos de características con atributos para pruebas en sistemas de alta variabilidad: primeros pasos. (Corto)

Leticia Montalvillo y Óscar Díaz: *Adaptando Github para el desarrollo de LPS: modelo de branching y operaciones de repositorio para los ingenieros del producto. SPLC* '15 Proceedings of the 19th International Conference on Software Product Line, ACM, 111-120, 2015. (Relevante)

Amador Durán, David Benavides, Sergio Segura, Pablo Trinidad y Antonio Ruiz-Cortés: *FLAME: a formal framework for the automated analysis of software product lines validated by automated specification testing*. Journal of Software & Systems Modeling (DOI: 10.1007/s10270-015-0503-z) (Relevante)

José A. Galindo, Hamilton A. Turner, David Benavides, Jules White: *Probando sistemas altamente configurables mediante análisis automático de modelos de características: El caso de Android.* Software Quality Journal 24(2): 365-405 (2016) (Relevante)

Javier González-Huerta, Emilio Insfrán, Silvia Abrahão, Giuseppe Scanniello: *Validación de un Método Dirigido por Modelos para la Evaluación y Mejora de Arquitecturas Software: Una Familia de Experimentos.* Information & Software Technology 57: 405-429 (2015) (Relevante)

Aplicando *scaffolding* en el desarrollo de Líneas de Producto Software*

Nieves R. Brisaboa, Alejandro Cortiñas, Miguel R. Luaces, Óscar Pedreira

Laboratorio de Bases de Datos
Universidade da Coruña
Campus de Elviña, A Coruña, España
{brisaboa, alejandro.cortinas, luaces, oscar.pedreira}@udc.es

Resumen Las Líneas de Producto Software (LPS) constituyen una tecnología madura para producir software que ha sido objeto de una gran cantidad de investigación, por lo que existen numerosas técnicas, metodologías y herramientas para crearlas. Sin embargo, es complicado utilizar algunas de estas herramientas en la industria debido a factores como la rápida evolución que han tenido los entornos de desarrollo, lo que provoca que estas herramientas estén obsoletas, la falta de soporte para proyectos que utilizan diferentes lenguajes de desarrollo, o la dificultad en el mantenimiento del código de los productos generados por la LPS. Por otra parte, la popularidad de la técnica de *scaffolding* no ha parado de aumentar entre los desarrolladores de software desde que apareció hace unos años, a pesar de recurrir a alternativas poco valoradas en la academia tales como el uso de preprocesadores.
En este trabajo proponemos la utilización de la técnica de *scaffolding* para implementar una LPS, lo que nos permite superar algunas de las limitaciones clásicas de otras herramientas LPS.

Palabras clave: ingeniería de líneas de producto software, *scaffolding*, arquitectura software de propósito general, gestión de la variabilidad, desarrollo de software orientado a características

1. Introducción

El desarrollo de software basado en Líneas de Producto Software (LPS) es un campo de investigación con casi dos décadas de antigüedad que ha producido multitud de metodologías y herramientas. A pesar de los grandes avances en el formalismo de las LPS y de las ventajas que proporcionan al proceso de desarrollo de software [1,2,3,4], su aplicación en las empresas de desarrollo de software se ha visto reducido a contextos muy específicos como el desarrollo de sistemas embebidos [5,6,7]. En otras áreas más complejas, como el desarrollo

* Financiado por el CDTI y el Ministerio de Economía y Competitividad (PGE y Fondos FEDER) [TIN2013-46238-C4-3-R, TIN2013-46801-C4-3-R, Ref. IDI-20141259, Ref. ITC-20151305]; y por la Xunta de Galicia (cofinanciado por FEDER) [GRC2013/053]

de aplicaciones web u otros ámbitos con tecnologías punteras y cambiantes, la aplicación de la ingeniería de Líneas de Producto Software se aplica cada vez con mayor frecuencia, aunque las propuestas en la literatura normalmente se centran en el modelado de la variabilidad del dominio a un nivel de abstración alto más que en la gestión de la variabilidad a nivel de la implementación [8]. Es por ello que existe la necesidad de nuevas herramientas y tecnologías LPS para este tipo de casos [9].

Paralelamente, en la industria de desarrollo de software han aparecido nuevas técnicas y metodologías para agilizar los procesos de desarrollo. Entre ellas destaca el *scaffolding*, popularizado por Ruby on Rails, Yeoman o Spring Roo. El *scaffolding* consiste en la generación de código a partir de plantillas predefinidas y de una especificación proporcionada por el desarrollador. Se suele utilizar para generar código repetitivo que se puede especificar fácilmente y generar a partir de una plantilla. De esa forma, el desarrollador se ahorra parte de la codificación ya que sólo tendrá que revisar y retocar mínimamente el código generado. Aunque el *scaffolding* es un concepto relativamente moderno en el desarrollo de software, los lenguajes de cuarta generación, de moda en los años 80, tenían funcionalidades similares (por ejemplo, generando formularios directamente a partir del modelo de datos de la base de datos). Igualmente, el *scaffolding* puede considerarse como una aplicación informal de los conceptos definidos en el campo de investigación de desarrollo de software dirigido por modelos.

Nuestro objetivo inicial fue crear una LPS para automatizar la generación de aplicaciones web para la gestión de información geográfica [10,11]. Sin embargo, a la hora de elegir la tecnología para implementarla no hemos encontrado ninguna alternativa en el contexto de las LPS que nos permita desarrollar la plataforma deseada debido a las siguientes cuestiones:

- Las herramientas LPS comerciales tienen un coste elevado que no puede ser asumido por una empresa de tamaño medio o pequeño.
- Las herramientas LPS no comerciales suelen depender de tecnología obsoleta (por ejemplo, una versión del entorno de desarrollo Eclipse de 2009 [12]).
- El código de los productos generados suele ser difícil de mantener por su complejidad y falta de legibilidad, especialmente con los enfoques compositivos. Hay muchos trabajos en los que se considera importante y se analiza la claridad del código de la plataforma [13,14], pero no hemos encontrado ninguno en el que se analice la claridad del código de los productos generados.
- Las herramientas LPS suelen estar orientadas a generar productos que usan un único lenguaje de programación (por ejemplo, Java) siendo difícil su aplicación si lo que se quiere generar son productos multilenguaje (por ejemplo, con un servidor Java y un cliente JavaScript).

Por otra parte, debido al diseño de nuestra herramienta, es necesario que parte del código del producto se genere desde cero en tiempo de desarrollo. Por ejemplo, el modelo de datos de nuestros productos será variable (específico de cada producto), por lo que el código fuente para el manejo de los datos tiene que ser generado a partir de la especificación que haga el analista (en concreto esto supone la generación de scripts SQL y de clases Java).

Por todo ello hemos desarrollado un motor de derivación para LPS basado en *scaffolding*. Nuestra propuesta, por una parte, tiene en cuenta conceptos formales de LPS tales como la gestión de un modelo de variabilidad y la validación de las características seleccionadas para un producto. Por otra parte, proporciona las ventajas de *scaffolding*, permitiendo generar productos multilenguaje basados en tecnologías modernas y ampliamente soportadas (HTLM, CSS,JavaScript, etc.) y, mediante el uso de plantillas previamente definidas y bien estructuradas, facilita la generación de código claro, legible y extensible a partir de la especificación del analista.

El resto del artículo se organiza de la siguiente forma: la Sección 2 describe el contexto que motiva la necesidad de nuestra propuesta, la Sección 3 describe el funcionamiento de la herramienta que implementa nuestra propuesta, la Sección 4 presenta un caso de uso concreto de la herramienta, la Sección 5 describe nuestras conclusiones respecto al uso de LPS o *scaffolding*, y la Sección 6 presenta las conclusiones y el trabajo futuro.

2. Contexto, motivación y objetivos

Para entender las decisiones que se han tomado en el diseño de nuestra solución para implementar LPS es necesario explicar el contexto en el que se utilizará. Los aplicaciones habituales que desarrolla la empresa *spin-off* asociada a nuestro grupo de investigación comparten un gran número de funcionalidades comunes (por ejemplo, suelen ser aplicaciones web que tienen gestión de usuarios con autenticación, páginas de contenido estático, listados y formularios de datos, importación y exportación de datos, generación de informes, visualización de información geográfica, etc.). Dado que las necesidades de los clientes son diversas, el modelo de datos es variable y los listados y formularios deben generarse en función del mismo. Además de las funcionalidades comunes, hay bastantes aplicaciones cuya casuística requiere desarrollos bastante específicos y, a priori, no aprovechables en ninguna otra aplicación.

En resumen, los desarrollos de la empresa *spin-off* comparten una serie de características comunes, y además cuentan con ciertas características particulares y exclusivas a cada uno de ellos. Lo que pretendemos es tener una herramienta de generación de código que de soporte al conjunto de características comunes y que genere productos fácilmente extensibles para incluir las funcionalidades específicas.

Teniendo en cuenta lo anterior, hemos confeccionado una serie de requisitos que debe cumplir la herramienta o tecnología usada para implementar nuestra LPS.

- *El código producido debe ser legible y extensible.* Algunas herramientas LPS como AHEAD o FeatureHOUSE, al generar métodos y clases artificiales para aplicar sus mecanismos de composición, producen un código final poco legible (se puede observar en la Figura 1 cómo se genera un método privado artificial al componer dos características que afectan al mismo método).

```
@RestController
@RequestMapping("/api/calculator")
public class CalculatorResource {

    @RequestMapping(method = RequestMethod.POST)
    private ResponseEntity<ResultJSON> calculate__wrappee__Base(@RequestBody RequestJSON
     request) {
        return response entity;
    }

    @RequestMapping(method = RequestMethod.POST)
    public ResponseEntity<ResultJSON> calculate(@RequestBody RequestJSON request) {
        if operation is division {...}
        return calculate__wrappee__Base(request);
    }
}
```

Figura 1. Ejemplo de código generado por FeatureHOUSE

Nuestra herramienta debe generar un código claro y legible para que sea posteriormente extensible. Al usar *scaffolding* la legibilidad del código va a depender de la calidad de las plantillas que usemos, y no se va a ver afectado por la utilización de la herramienta en sí.

- *Los productos producidos deben poder adaptarse a cualquier diseño.* Es decir, puesto que se trata de desarrollar productos diferentes para clientes diferentes, nuestra herramienta debe producir un código fuente que se pueda extender fácilmente dándole al equipo de desarrollo total libertad en el diseño del producto final, evitando el imponerle estructuras rígidas e inmodificables.
- *La herramienta no debe imponer tecnologías o entornos de trabajo.* Nuestra herramienta producirá código basado en plantillas adaptadas a los entornos de trabajo habituales de la *spin-off*.
- *La herramienta debe ser fácil de usar.* Se trata de que el coste de formación en el uso de la misma sea mínimo.
- *La herramienta debe producir código multilenguaje.* Todos los proyectos de la *spin-off* incluyen una mezcla de varias lenguajes de programación, entre otros: SQL, Java, JSP, JSTL, C#, Visual Basic, ASP, HTML, CSS, y JavaScript. Pocas herramientas LPS existentes en el estado del arte [2,3,4] producen código en diferentes lenguajes de programación a la vez.

Estos requisitos hacen inviable para nuestros propósitos la utilización de la mayor parte de herramientas LPS existentes.

3. Arquitectura, implementación y funcionamiento de la herramienta

3.1. Arquitectura

La Figura 2 muestra la estructura de la herramienta de derivación de la LPS mediante un diagrama de componentes de UML. El componente central es el

motor de derivación que se encarga de gestionar todo el proceso de generación del producto. Para realizar esta tarea delega en tres componentes: el *gestor del modelo de variabilidad*, el *motor de plantillas*, y el *gestor de ficheros*. El componente de la parte superior de la figura es un *cliente de línea de comandos* que proporciona un acceso sencillo al usuario analista para generar nuevos productos.

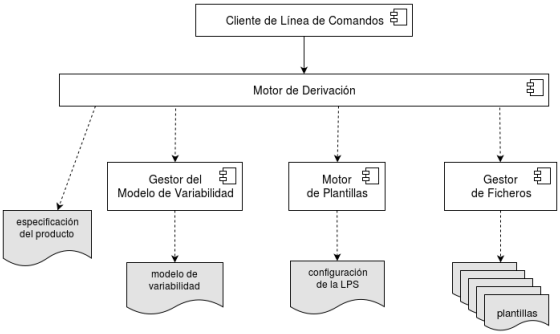


Figura 2. Diagrama de componentes de nuestra herramienta

El ***Gestor del Modelo de Variabilidad*** se encarga de construir el modelo de características a partir de la especificación del usuario y proporciona funcionalidades para importar/exportar un modelo de variabilidad desde y hacia un fichero, para validar que el modelo de variabilidad sea correcto (por ejemplo, que no tenga ninguna característica hoja que sea abstracta), para validar la selección de características que realiza el analista a la hora de generar los productos, y para detectar posibles problemas en el código fuente anotado (por ejemplo, características que aparecen en anotaciones pero no aparecen definidas en el modelo de variabilidad, o viceversa).

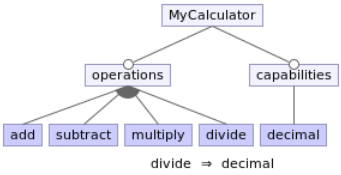


Figura 3. Modelo de características de una calculadora web

El modelo de variabilidad se indica siguiendo la especificación de Feature IDE (en el formato XML de Feature IDE, en JSON o en YAML) y permite restricciones con operadores lógicos además del árbol de variabilidad clásico. En la Figura 3 mostramos un modelo de variabilidad de una LPS simple que genera calculadoras web con distintas funcionalidades.

```
1  @RequestMapping(method = RequestMethod.POST)
2  public ResponseEntity<ResultJSON> calculate(@Valid @RequestBody RequestJSON request) {
3      /*% if (feature.divide) { %*/
4      if (request.getOperation() == Operation.DIVIDE) {
5          if (request.getSecond() == 0) {
6              return new ResponseEntity<ResultJSON>(new ResultJSON(), HttpStatus.BAD_REQUEST);
7          }
8      }
9      /*% } %*/
10     return new ResponseEntity<ResultJSON>(new
        ResultJSON(calculatorService.calculate(request.getFirst(),
11             request.getSecond(), request.getOperation())), HttpStatus.OK);
12 }
```

Figura 4. Ejemplo de método Java anotado

El ***Motor de Plantillas*** utiliza técnicas de *scaffolding* para aplicar la variabilidad seleccionada a las plantillas de la LPS y generar el código del producto final a partir del código fuente anotado y la especificación de la LPS. Las anotaciones en las plantillas se indican mediante comentarios del lenguaje de programación de la plantilla y su contenido es cualquier código JavaScript. En la Figura 4 se muestra un método de una clase Java anotado. En la línea 3 se indica que las líneas 4 a 8 sólo se deben incluir en el código del producto final si el analista ha seleccionado la característica *divide*. Como vemos, la anotación se ha realizado dentro de un comentario Java por lo que no interfiere con el compilador, IDE, o herramienta de validación que esté usando el desarrollador de la LPS.

Además de anotaciones relacionadas con las características, el motor de plantillas permite usar variables en las plantillas. En la Figura 5 se muestra un fichero de definición de proyecto de Maven con variables para la identificación del proyecto que serán sustituidas en el producto por los valores indicados por el analista durante el proceso de generación del producto. Finalmente, el motor de plantillas también puede validar la especificación y las plantillas con funcionalidades como detectar qué características no están referenciadas en ninguna anotación, detectar qué anotaciones no están definidas en el modelo de características, o calcular la complejidad de cada característica analizando cuanto código se ve modificado por cada una de ellas.

Por último, el ***Gestor de Ficheros*** se encarga de encapsular todas las tareas de acceso a las plantillas y permite recorrer recursivamente las carpetas en las que se encuentran las plantillas, leer y escribir ficheros de texto, detectar los

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <groupId><!-- %= data.maven.group %--></groupId>
  <artifactId><!-- %= data.maven.artifact %--></artifactId>
  <version><!-- %= data.maven.version %--></version>
  <packaging>war</packaging>
  <name><!-- %= data.title %--></name>
  <description><!-- %= data.description %--></description>
</project>
```

Figura 5. Ejemplo de variables en fichero de proyecto Maven

ficheros binarios para no procesarlos y copiarlos directamente en el directorio de destino, etc.

3.2. Implementación

Desde el punto de vista tecnológico, nuestra solución está implementada en Node.js, un entorno de ejecución multiplataforma basado en el motor JavaScript V8 de Google Chrome. Node.js es ligero, independiente del sistema operativo y de cualquier IDE y su uso está muy extendido y en continuo crecimiento. La librería sólo requiere tener Node.js instalado, que está disponible para todos los sistemas operativos y que además es una plataforma muy activa y en crecimiento, por lo que se espera que su uso siga vigente durante muchos años.

Hasta llegar a la decisión de diseñar e implementar nuestro propio motor de derivación hemos desechado algunas alternativas. La primera aproximación fue utilizar Yeoman[1], una conocida plataforma de *scaffolding* que permite implementar *generadores de arquitecturas* de proyectos de cualquier tipo. Sin embargo, Yeoman tenía dos problemas: es una plataforma demasiado amplia y compleja para nuestras necesidades, y además exige programación *ad hoc* para cada proyecto en el que es usada. El siguiente paso fue utilizar únicamente el motor de plantillas de Yeoman, y otros motores de plantillas modernos. Finalmente acabamos descartando todos ellos por dos motivos: por un lado, ninguna de ellas cubría totalmente nuestros requisitos en cuanto a gestión de las anotaciones, y por otro lado ninguna proporciona funcionalidad para analizar el código anotado respecto al modelo de variabilidad.

Siguiendo este enfoque hemos diseñado e implementado un motor de derivación de LPS basado en *scaffolding* que cumple los requisitos descritos en la Sección 2 y además tiene las siguientes características:

- *Los componentes reutilizables de la LPS son las plantillas del* scaffolding. Los desarrolladores de la LPS usan las anotaciones del *scaffolding* para indicar los bloques que se corresponden con cada característica de la LPS. Esto no

[1] http://yeoman.io/

se aleja demasiado del uso de tecnologías anotativas para implementar LPS, como Antenna [15], Munge [16] o cualquier preprocesador.

- *La especificación del* scaffolding *se realiza con el modelo de variabilidad de la LPS.* La herramienta de *scaffolding* recibe como especificación el modelo de variabilidad y la selección de características del analista. Esta especificación se usa para validar la selección de características realizada y para generar el código fuente del producto a partir de las plantillas.
- *Anotaciones no intrusivas.* Dado que se usa un enfoque anotativo y los componentes se consideran plantillas de código, el lenguaje utilizado no es una limitación. Además, las anotaciones se realizan con comentarios del lenguaje de programación lo que facilita el desarrollo de los componentes reutilizables al no impedir el uso de herramientas de desarrollo estándar.
- *La herramienta de* scaffolding *permite generación de código a partir de la especificación.* Más allá de las funcionalidades estándar de una LPS de generación de código final mediante la inclusión y/o eliminación de bloques de código, nuestra herramienta además está capacitada para generar código fuente desde cero a partir de la especificación. Además del modelo de variabilidad y la selección realizada por el diseñador, la LPS puede recibir especificaciones adicionales (por ejemplo, el modelo de datos o la configuración del aspecto visual del producto) y la herramienta de *scaffolding* puede generar la funcionalidad de acceso a datos o las hojas de estilo CSS a partir de plantillas completadas con la especificación completada. También se permite en las plantillas utilizar bucles y otras estructuras de control avanzadas para generar código, lo que da al diseñador de la LPS una gran libertad y flexibilidad.

3.3. Funcionamiento

El proceso de generación de un producto se invoca utilizando el cliente de terminal y requiere cinco parámetros: el fichero con el modelo de variabilidad, el fichero con la configuración de la LPS, el fichero con la especificación del producto a generar, la carpeta con las plantillas de la LPS, y la carpeta donde se generará el producto nuevo.

Al recibir esos cinco parámetros, el motor de derivación comienza delegando en el gestor del modelo de variabilidad la interpretación del modelo proporcionado por el usuario. Si es correcto, el siguiente paso consiste en interpretar el fichero de configuración de la LPS que indica aspectos como qué cadenas de texto delimitarán el inicio y el fin de las anotaciones en función de la extensión de los ficheros a procesar, o qué ficheros de la carpeta de plantillas se deben ignorar a la hora de generar productos. Esta funcionalidad permite que elementos usados en el desarrollo de los componentes reusables no se transfieran al producto final.

A continuación, el motor de derivación interpreta el fichero con la especificación del producto a generar. Este fichero incluye la selección de características no abstractas que el analista desea para el producto, y la colección de valores que se usarán para sustituir las variables referenciadas en las anotaciones del código fuente. En la Figura 6 vemos un ejemplo de especificación de producto en la que

```
features:
  - decimal
  - add
  - subtract
  - multiply
  - divide
```

```
data:
  title: The Calculator
  description: A simple web calculator
  maven:
    group: es.udc.lbd
    artifact: web-calculator
    version: 0.1
```

Figura 6. Ejemplo de especificación del producto a generar

en el objeto *features* se enumeran las características del modelo de variabilidad de la Figura 3 que se desean para la calculadora concreta que se quiere producir, y en el objeto *data* se indican los valores usados para sustituir las variables de la plantilla de la Figura 5.

En el último paso del proceso, el motor de derivación delega en el motor de plantillas y el gestor de ficheros para generar el código fuente del producto. Para ello, el motor de plantillas se instancia con la configuración específica de la LPS y comienza a recorrer todos los ficheros de la carpeta de plantillas procesando las anotaciones. Este procesamiento se realiza generando para cada plantilla una función JavaScript que, en caso de encontrar anotaciones evalúa el contenido de la misma, y en caso encontrar texto sin anotaciones lo copia directamente al fichero salida. La evaluación de esta función JavaScript genera en la carpeta del producto un fichero de código fuente con las anotaciones aplicadas.

4. Caso de estudio

La *Graph Product Line* (GPL) es una LPS de algoritmos de grafos propuesta por [17] como un problema estándar para la evaluación de tecnologías LPS. GPL consiste en 15 características relacionadas con los grafos en sí mismos, y 2 características relacionadas con su ejecución. Además, las implementaciones de GPL suelen llevarse a cabo de 3 formas diferentes, como también se describe en [17]. Si bien GPL es una LPS simple, es un buen método de evaluación ya que tiene cierta complejidad difícil de abordar aún diseñándolo desde el principio como una LPS, especialmente cuando se mezclan diferentes configuraciones. Las técnicas que no permiten implementar variabilidad con un nivel de granularidad muy fino requieren hacer ciertas concesiones a la hora de implementar GPL tales como replicar bastante código en varias características, o diseñar la LPS con bastantes características y restricciones artificiales.

En cuanto a nuestra implementación, nos hemos centrado principalmente en la simplicidad y legibilidad del código, y por tanto hemos reimplementado desde cero todo el problema en vez de aprovechar código de implementaciones existentes. Además, hemos encontrado bastantes problemas en las implementaciones que hemos podido analizar, como variantes de productos que no llegaban ni siquiera a compilar por errores en el código generado.

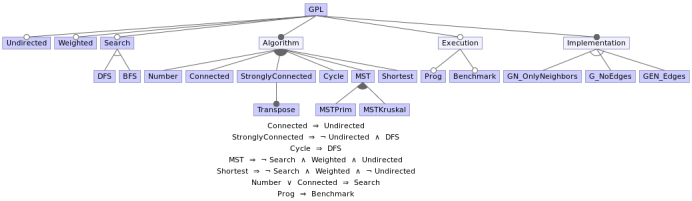


Figura 7. Modelo de características de GPL

Nuestro modelo de variabilidad, que podemos ver en la Figura 7, cuenta con 23 características, siendo 20 de ellas no abstractas, es decir, que tienen influencia en algún fichero de código fuente, y siendo 16 de ellas características hoja. Nuestra herramienta permite, al contrario que otras alternativas, tener características no abstractas que al mismo tiempo no sean hoja, ya que es útil en ciertos casos, por ejemplo para anotar el código común a las características *DFS* y *BFS*, que comparten la característica padre no abstracta *Search*. No existen las características artificiales que vemos en otras alternativas, dado que nuestra herramienta es suficientemente flexible como para proporcionar la granularidad fina necesaria para permitir las tres implementaciones sin requerir restricciones o características artificiales. Podemos ver un ejemplo de esto en el código 8, donde vemos la variación en función de la implementación. Para finalizar, tenemos 7 restricciones que afectan a 12 características.

```
1  graph.getVertices().forEach(v -> {
2
3    /*% if (feature.gNoEdges) { %*/
4    v.getAdjacents().forEach(adj -> edges.add(new Edge(v, adj, v.getWeight(adj.getName()))));
5    /*% } else if (feature.gnOnlyNeighbors) { %*/
6    v.getNeighbors().forEach(nei -> edges.add(new Edge(v, nei.getNeighbor(),
       nei.getWeight())));
7    /*% } else if (feature.genEdges) { %*/
8    v.getNeighbors().forEach(n -> edges.add(n.getEdge()));
9    /*% } %*/
10
11   sets.put(v, new HashSet<IVertex>());
12   // each vertex should be a disjoint set at first
13   sets.get(v).add(v);
14 });
```

Figura 8. Anotación en el código de GPL

Cuando nos comparamos con alternativas compositivas como AHEAD o Feature House quedan patentes los problemas del enfoque compositivo a la hora de implementar la variabilidad *fine grained*. El número de características y res-

tricciones aumenta artificialmente en ambos casos. En concreto, la versión de AHEAD cuenta con 66 características y 32 restricciones (afectando a 41 de las características), mientras que en la versión de Feature House encontramos 38 características y 16 restricciones (afectando a 24 características en este caso). Este mismo problema fue señalado por Kästner et al. [18], cuyo modelo de variabilidad para implementar GPL se acerca mucho más al nuestro, contando con 29 características y 8 restricciones (afectando a 18 características). Parte de las ventajas de nuestra aproximación vienen dadas directamente por la posibilidad de usar las últimas versiones de Java, ya que la ausencia de interfaces, por ejemplo, en el código de AHEAD, es una gran tara.

5. Lecciones aprendidas: Líneas de Producto Software vs *scaffolding*

Hay una clara similitud entre el proceso *scaffolding* y el funcionamiento de una línea de producto software. En ambos casos se parte de ciertos activos de los que, en función de una determinada especificación, se obtiene una salida en forma de código fuente. Los activos son las plantillas anotadas o los componentes reusables, la especificación es una descripción del componente a construir o la selección de características, y la salida se obtiene mediante generación de código o a través del montaje de los distintos componentes. Sin embargo, hay algunas diferencias significativas:

- Generalmente, en una LPS las características se implementan con bloques de código completos que se incluyen o no en los productos finales. En cambio, en *scaffolding* se genera código fuente, ya sea en tiempo de compilación o en tiempo de ejecución.
- Los productos que se generan en una LPS normalmente son productos finales listos para pasar a producción mientras que las herramientas de *scaffolding* están orientadas a ser una ayuda para los desarrolladores.
- En el caso de las LPS es un analista el que genera los productos, dado que simplemente debe elegir las características deseadas entre una lista. En el caso de *scaffolding* suele ser el desarrollador en el que utiliza las herramientas.
- El código fuente generado por las herramientas de *scaffolding* es claro y fácilmente mantenible porque está orientado a desarrolladores, mientras que el código fuente del producto de una LPS acostumbra a ser difícilmente mantenible porque no se espera que sea modificado.
- Las herramientas de *scaffolding* están muy presentes en la comunidad de desarrollo de software y su evolución y uso es muy activo, mientras que las tecnologías de LPS se usan mucho en determinados sectores pero no de manera generalizada.

Después de analizar las diferentes técnicas para implementar LPS, listadas en [2,3,4], hemos detectado una serie de problemas comunes:

- *Lejanía respecto a los lenguajes de desarrollo actuales.* Por una parte, aunque existen herramientas LPS que permiten realizar composición sobre varios lenguajes de programación al mismo tiempo (por ejemplo, FeatureHouse [19] o CIDE [18]), en la práctica sólo se puede utilizar directamente un conjunto muy pequeño puesto que hay que implementar extensiones particulares para cada lenguaje. Por otra parte, algunas de las herramientas sólo funcionan con determinadas versiones del lenguaje de programación en cuestión (por ejemplo, no hemos encontrado ninguna alternativa compositiva que permita usar toda la funcionalidad de las anotaciones de Java disponibles desde la versión 1.5 de 2004).
- *Lejanía respecto a las herramientas de desarrollo actuales.* Muchas tecnologías de implementación de LPS obligan a utilizar herramientas de desarrollo concretas, algunas de ellas ya obsoletas. Es bastante común el uso de la plataforma Eclipse, pero no siempre las herramientas funcionan con la última versión de la misma. Por ejemplo, la herramienta CIDE funciona con Eclipse 3.5 que fue publicado en 2009. En otros casos, el problema se presenta en el momento de hacer funcionar la herramienta. Por ejemplo, hacer funcionar preprocesadores de propósito general (GPP o GNU M4) es realmente complicado.
- *El código del producto generado no es fácilmente mantenible.* El código fuente de los productos generados por las enfoques compositivos acostumbra a no ser mantenible, como ya hemos comentado en la Sección 2. Esto no es un problema en contextos en los que el producto generado por la LPS se despliega sin necesitar cambios, pero en los casos donde no es así modificar el comportamiento de código de un producto generado por una LPS compositiva es una tarea muy complicada e imposible de asumir para un equipo de desarrollo.

Las técnicas de *scaffolding* solucionan estos problemas: están orientadas a las técnicas de desarrollo actuales dando soporte a múltiples lenguajes de programación y funcionando con múltiples entornos de desarrollo, y el código fuente generado es fácilmente legible y mantenible. Sin embargo, también presentan problemas:

- *Falta de formalismo.* Cada tecnología de *scaffolding* define un modelo *ad-hoc* para especificar los componentes que se generan.
- *Elevada complejidad en la definición de las plantillas.* Muchos autores se posicionan en contra del uso de preprocesadores por la problemática de las anotaciones *fine-grained* [20,21] que hacen extremadamente complejo el mantenimiento del código de las plantillas. Pese a ello, los preprocesadores se mantienen como la alternativa más popular en el ámbito industrial para implementar LPS [5,22,23].

Nuestra propuesta se sitúa en un punto intermedio entre los dos campos: usamos *scaffolding* para implementar Líneas de Producto Software de forma que solventemos ciertos problemas en las técnicas de implementación de LPS actuales y haciendo más atractiva su adopción para los equipos de desarrollo de software, pero sin perder los formalismos y conceptos detrás del desarrollo de LPS.

6. Conclusiones y trabajo futuro

En este artículo hemos mostrado nuestra propuesta para crear un motor de derivación LPS basado en *scaffolding*. Nuestra herramienta, por una parte, se apoya en los formalismos de líneas de producto software, y por otra parte proporciona las ventajas de *scaffolding* al generar código fuente fácilmente mantenible y permitir utilizar tecnologías modernas y ampliamente soportadas. En el artículo se describen las decisiones de diseño que se tomaron para construir el motor de derivación, su arquitectura, y el proceso en el que se basa su funcionamiento. Además, se muestra un caso de estudio con una LPS de cierta complejidad en el que se ve que nuestra propuesta es mejor que otras alternativas en términos de complejidad de la LPS y de calidad del código del producto resultante.

Como trabajo futuro tenemos planificado mejorar la funcionalidad de análisis del modelo de variabilidad y del código fuente anotado para obtener mejores indicadores de la calidad del código de la LPS, mejorar la generación de código para permitir la generación de múltiples ficheros a partir de una plantilla y no sólo la generación de un fichero por plantilla con el mismo nombre, y estudiar la integración del motor de derivación en procesos de desarrollo de software permitiendo que las modificaciones en los componentes de la LPS se trasladen fácilmente a los productos generados.

Referencias

1. Pohl, K., Böckle, G., Linden, F.J.v.d.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2005)
2. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines: Concepts and Implementation. Springer-Verlag, Berlin/Heidelberg (2013) 308 pages, ISBN 978-3-642-37520-0.
3. Meinicke, J., Thüm, T., Schröter, R., Benduhn, F., Saake, G.: An overview on analysis tools for software product lines. In: Workshop on Software Product Line Analysis Tools (SPLat), New York, NY, USA, ACM (September 2014) 94–101
4. Databases and Software Engineering Workgroup — University of Magdeburg: Tools for feature-oriented software development. `http://wwwiti.cs.uni-magdeburg.de/iti_db/research/fosd-tools/` (Consultado el 02/07/2016).
5. Berger, T., Rublack, R., Nair, D., Atlee, J.M., Becker, M., Czarnecki, K., Wasowski, A.: A survey of variability modeling in industrial practice. In: The Seventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '13, Pisa , Italy, January 23 - 25, 2013. (2013) 7:1–7:8
6. Weiss, D.M.: The product line hall of fame. In: Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings. (2008) 395
7. van der Linden, F.: Software product families in europe: The esaps & café projects. IEEE Software **19**(4) (2002) 41–49
8. do Carmo Machado, I., Santos, A.R., Cavalcanti, Y.C., Trzan, E.G., de Souza, M.M., de Almeida, E.S.: Low-level variability support for web-based software product lines. In: The Eighth International Workshop on Variability Modelling of

Software-intensive Systems, VaMoS '14, Sophia Antipolis, France, January 22-24, 2014. (2014) 15:1–15:8
9. Urli, S., Blay-Fornarino, M., Collet, P.: Handling complex configurations in software product lines: a tooled approach. In: 18th International Software Product Line Conference, SPLC '14, Florence, Italy, September 15-19, 2014. (2014) 112–121
10. Brisaboa, N.R., Cortiñas, A., Luaces, M.R., Pol'la, M.: A Reusable Software Architecture for Geographic Information Systems based on Software Product Line Engineering. In: Proceedings of the 5th International Conference on Model & Data Engineering (MEDI 2015), Springer (2015) 320–331
11. Brisaboa, N.R., Cortiñas, A., Luaces, M.R., Pol'la, M.: Gisbuilder: a framework for the semi-automatic generation of web-based geographic information systems. In: Proceedings of the 20th Pacific Asia Conference on Information Systems (PACIS 2016). (2016) (Pendiente de publicación).
12. Kästner, C.: Cide: Virtual separation of concerns. `http://wwwiti.cs.uni-magdeburg.de/iti_db/research/cide/` (Consultado el 02/07/2016).
13. Kästner, C., Apel, S., Kuhlemann, M.: Granularity in software product lines. In: 30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008. (2008) 311–320
14. Feigenspan, J., Kästner, C., Apel, S., Liebig, J., Schulze, M., Dachselt, R., Papendieck, M., Leich, T., Saake, G.: Do background colors improve program comprehension in the #ifdef hell? Empirical Software Engineering **18**(4) (2013) 699–745
15. Pleumann, J., Yadan, O., Wetterberg, E.: Antenna. an ant-to-end solution for wireless java. `http://antenna.sourceforge.net/` (Consultado el 27/04/2016).
16. Munge Development Team: Munge: Simple Java preprocessor. `https://github.com/sonatype/munge-maven-plugin` (Consultado el 27/04/2016).
17. Lopez-Herrejon, R.E., Batory, D.S.: A standard problem for evaluating product-line methodologies. In: Generative and Component-Based Software Engineering, Third International Conference, GCSE 2001, Erfurt, Germany, September 9-13, 2001, Proceedings. (2001) 10–24
18. Kästner, C., Apel, S., Trujillo, S., Kuhlemann, M., Batory, D.S.: Guaranteeing syntactic correctness for all product line variants: A language-independent approach. In: Objects, Components, Models and Patterns, 47th International Conference, TOOLS EUROPE 2009, Zurich, Switzerland, June 29-July 3, 2009. Proceedings. (2009) 175–194
19. Apel, S., Kästner, C., Lengauer, C.: FEATUREHOUSE: language-independent, automated software composition. In: 31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings. (2009) 221–231
20. Spencer, H., Collyer, G.: #ifdef considered harmful, or portability experience with C news. In: USENIX Summer 1992 Technical Conference, San Antonio, TX, USA, June 8-12, 1992. (1992)
21. Favre, J.: Understanding-in-the-large. In: 5th International Workshop on Program Comprehension (WPC '97), May 28-30, 1997 - Dearborn, MI, USA. (1997) 29–38
22. Liebig, J., Apel, S., Lengauer, C., Kästner, C., Schulze, M.: An analysis of the variability in forty preprocessor-based software product lines. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010. (2010) 105–114
23. Kästner, C., Apel, S.: Virtual separation of concerns - A second chance for preprocessors. Journal of Object Technology **8**(6) (2009) 59–78

Estudio del Soporte a la Variabilidad en la Nube en un entorno con Multitenencia: Plataforma GPaaS

Héctor Humanes, Iván Hernández, Jessica Díaz, Jennifer Perez

Universidad Politecnica de Madrid (UPM)
CITSEM
Madrid, Spain
yesica.diaz@upm.es,
jenifer.perez@etsisi.upm.es,

Alfonso Ríos, Javier Gonzalez-Rodriguez, Jordi Paraire

Minsait, Indra
Valencia, Spain
ariosa@minsait.com,
jvgonzalez@minsait.com,
jjparaire@minsait.com

Abstract. Los requisitos de la sociedad actual y la nueva era de Internet de las Cosas (*Internet of Things*, IoT), entre otros múltiples factores, explican el auge del software como servicio (*Software as a Service*, SaaS)) y el paradigma de computación en la nube (*Cloud Computing*). La tendencia en el desarrollo software apunta hacia la producción de software cada vez más flexible, dinámico y personalizado, que a su vez, es accesible a través de Internet (*off-premises*), sin necesidad de ser instalado y gestionado localmente (*on-premises*). Una de las propiedades clave de *Cloud Computing* es la multitenencia: la instanciación de varias ocurrencias software a partir de una aplicación base o recursos compartidos. En este artículo se presenta: (i) un estudio de la multitenencia y el soporte a la variabilidad en la nube; y (ii) una experiencia de desarrollo SaaS cuyo objetivo es analizar la capacidad de la multitenencia para soportar la flexibilidad, adaptabilidad y variabilidad del software en la nube, así como sus limitaciones, con el fin identificar líneas de investigación futuras. En particular, el estudio y análisis se ha realizado en los laboratorios de la iSmart Software Factory (iSSSF) de la Universidad Politécnica de Madrid (UM) utilizando la plataforma en la nube de Minsait (Indra), llamada GPaaS,

Keywords. Computación en la Nube (*Cloud Computing*), Multitenencia, Variabilidad, Flexibilidad, Personalización Masiva, Adaptabilidad, Reconfiguración Dinámica

1 Introducción

En los últimos años, nuestra sociedad está cambiado a gran velocidad, ya que cada día más, estamos rodeados de dispositivos inteligentes y permanentemente estamos conectados a Internet mediante ordenadores, tabletas, o móviles. Esta nueva era de Internet de las Cosas (*Internet of Things – IoT*) y de servicios al ciudadano a través de internet, con la nueva concepción de los sistemas inteligentes (*smart buildings, grids, cities* y *spaces*), explican el auge del software como servicio (*Software as a Service*, SaaS)) y el paradigma de computación en la nube (*Cloud Computing*).

Cloud Computing es un paradigma que nos permite ofrecer servicios a través de una red, normalmente Internet, y que son accesibles al usuario sin tener que gestionar los recursos que ofrecen tales servicios [18]. Este paradigma se fundamenta en la

virtualización, reutilización y compartición de recursos, y al pago por uso de estos recursos, ofreciendo características clave [18] como: agilidad (autoprovisionamiento), independencia de localización, multitenancia, fiabilidad, escalabilidad y mantenibilidad. Por ello, *Cloud Computing* se presenta como un claro candidato a dar solución a las necesidades software de la sociedad actual, ya que la tendencia en el desarrollo software apunta hacia la producción de software cada vez más flexible, dinámico y personalizado, que a su vez, es accesible a través de Internet (*off-premises*), sin necesidad de ser instalado y gestionado localmente (*on-premises*).

La multitenencia que ofrece la computación en la nube consiste en la instanciación de varias ocurrencias software a partir de una aplicación base o recursos compartidos. Esta propiedad ofrece la posibilidad de gestionar la variabilidad, y por ende la flexibilidad en el desarrollo en la nube. Por ello, este artículo presenta un estudio de la multitenencia y el soporte a la variabilidad en la nube. Para realizar dicho estudio se ha seleccionado la solución de la nube de Minsait (by Indra) (GPaaS [12]) ya que ofrece soporte a la multitenencia e independencia del proveedor de servicios sobre el que desarrollar la multitenancia, es decir, podemos desarrollar una aplicación multitenant y desplegarla en múltiples proveedores de la nube, como *Microsoft Azure*, *Google Cloud Platform*, *Amazon Web Services*, etc. Esta característica de abstracción fue la razón de evaluar GPaas y su soporte multitenant para gestionar variabilidad independientemente de la plataforma, frente a evaluar el soporte multitenant de plataformas específicas de otros proveedores de la nube. Dado que la finalidad de este estudio es identificar líneas de investigación futuras, se decidió realizar un caso de estudio entre la UPM y Minsait (by Indra) cuyo objetivo era analizar la capacidad y limitaciones de la multitenencia para soportar la flexibilidad y variabilidad del software en la nube, así *como* la reconfiguración dinámica de las aplicaciones. En este artículo presentamos el diseño del caso de estudio, así como los resultados obtenidos. El resto del artículo está organizado con la estructura que se detalla a continuación. La sección 2 presenta los conceptos previos sobre variabilidad, personalización masiva, y la computación de la nube, así como el estudio al soporte de la multitenencia en la nube. La sección 3 presenta las principales características y razones que han hecho que se elija la solución en la nube GPaaS para este estudio. La sección 4 detalla tanto el desarrollo del caso de estudio como los resultados obtenidos. Finalmente, la sección 5 presenta las conclusiones y los trabajos futuros.

2 Personalización Masiva y Variabilidad en la Nube

2.1 Variabilidad y Personalización Masiva

La variabilidad en el contexto software se puede definir como la habilidad de un producto software para cambiar y ser usado en múltiples contextos. El desarrollo de software flexible radica en el uso de la variabilidad, reflejado en el código fuente y establecido previamente en la arquitectura. El diseño basado en la flexibilidad (*designing for flexibility)* implica el diseñar las arquitecturas software como un conjunto de decisiones de diseño implementadas a través de mecanismos de soporte a la variabilidad [3]. La implementación de la variabilidad se puede categorizar entre variabilidad interna y externa [11]:

- **Variabilidad interna**: Se define como el conjunto de variantes, normalmente tecnológicas o de mejora del software, que pueden formar parte de un producto y que se caracterizan por no ser percibidas por el usuario final de la aplicación. Ejemplos de variabilidad interna son las bases de datos (BBDD) o los canales de intercambio de datos.
- **Variabilidad externa**: Se define como el conjunto de variantes que pueden ser percibidas por el usuario final del software. Ejemplos de variabilidad externa son las diferencias funcionales y de interfaz gráfica de usuario (IGU).

El software provisto en la nube necesita la personalización de productos mediante la producción de servicios de aplicaciones ajustadas a las necesidades individuales de cada consumidor o conjunto de consumidores (tenant), no sólo de las Infraestructuras (Infrastructure as a Services (IaaS)) utilizadas, sino también de las Plataformas de Desarrollo (Platform as a Service (PaaS)) y de los Servicios (Software as a Service (SaaS)). Dicha personalización no puede ser entendida sin haber dotado al software de mecanismos para soportar un alto grado de variabilidad. De forma que el desarrollo software se realice teniendo en cuenta las siguientes consideraciones:

- Inversión de la necesidad: Se rompe con el patrón tradicional del consumidor eligiendo aquel producto que más se ajusta a sus necesidades de entre los existentes. Es ahora el usuario quien indica qué necesita y el software se adapta a esa necesidad.
- Flexibilidad en las infraestructuras y plataformas de desarrollo: es posible personalizar el entorno en el que se ejecutará el software, lo que facilita la configuración y adaptación de entornos de trabajo.
- Soporte a requisitos volátiles o altamente cambiantes: Si se dota al software de una alta capacidad de personalización, también es posible responder rápidamente a cambios en los requisitos o necesidades identificadas a priori.
- Mejora del *time-to-market*: Configurar producto software es más rápido que desarrollar ese producto desde cero para un cliente.

2.2 Computación en la nube (*Cloud Computing)*

Cloud Computing es un paradigma que nos permite ofrecer servicios a través de una red, normalmente Internet, y que son accesibles al usuario sin tener que gestionar los recursos que ofrecen tales servicios. En los últimos años, la computación en la nube se está extendiendo de forma que el software se está desarrollando para ser implantado en la nube o se están buscando soluciones para migrar aplicaciones a la nube, y es que según Wang et al. [18] ‹‹*En 2020, la mayoría de las personas accederán a las aplicaciones software vía online y de forma compartida, y accederán a la información a través del uso de servidores remotos de la red; en lugar de depender principalmente de herramientas e información localizada en sus ordenadores personales*››. De cara al usuario, este modelo se basa en un acceso bajo demanda a una fuente de recursos computacionales configurables y al pago por su uso exclusivamente (*pay-per-use).* Tales recursos incluyen redes, servidores, almacenamiento, aplicaciones, plataformas y servicios que pueden ser rápidamente iniciados y estar accesibles en un breve lapso de tiempo sin una intervención explícita del proveedor del servicio.

La computación en la nube es el paradigma que nos ofrece todo como servicio *X as a Service (XaaS,* ofreciendo tres modalidades de servicio que siguen una arquitectura por capas, en orden ascendente:

- **Infraestructura como Servicio (IaaS)**: se ofrece un espacio de almacenamiento o capacidad de procesamiento en sus servidores, ilimitados en apariencia para el usuario y que dependerá de su poder adquisitivo para añadir opciones al servicio. El IaaS ofrece un servicio de *hosting* (ej. Amazon Web Services).
- **Plataforma como Servicio (PaaS):** se ofrecen distintas herramientas para la realización de desarrollos informáticos, de forma que se los usuarios puedan construir y desplegar sus aplicaciones sin necesidad de adquirir e implantar en sus ordenadores personales dichas herramientas. El PaaS ofrece un servicio de construcción (ej. Microsoft Azure, Google App Engine, etc).
- **Software como Servicio (SaaS):** se ofrecen aplicaciones finales como servicios. Dichas aplicaciones se encuentran situadas en los servidores del proveedor de servicios en la nube [5]. El SaaS ofrece un servicio de consumo o uso (ej. Google Drive).

Entre las ventajas que nos ofrece la computación en la nube son destacables: (i) la pequeña o nula inversión por parte del usuario en adquisición y mantenimiento del hardware que soporta los servicios que requiere, (ii) la inmediata escalabilidad y adaptabilidad al consumo de recursos hardware o software, (iii) la variedad y cantidad de aplicaciones existentes, (iv) la posibilidad de acceder a las aplicaciones o servicios desde cualquier lugar, y (v) el usuario paga exclusivamente por aquellos recursos que usa y cuando los usa. Wang et al. [18] definen el conjunto de características clave de la computación en la nube como agilidad (autoprovisionamiento), independencia de localización, multitenencia, fiabilidad, escalabilidad y mantenibilidad. En este artículo nos centramos en el estudio de la multitenencia como soporte a la variabilidad.

Multitenencia y Variabilidad en la Nube.

La multitenencia se basa en el concepto de tenant. Las aplicaciones orientadas a servicio tradicionales típicamente dedicaban una aplicación por tenant. Un tenant se puede definir como un conjunto de usuarios (ej. una organización) que comparten una serie de requisitos y necesidades. Tradicionalmente, los tenants pagan por el uso del producto software, del cual se crea una instancia personalizada en base a sus necesidades. Sin embargo, hoy en día los proveedores de SaaS tienden a adoptar una arquitectura *multitenant* [9], ya que todas estas instancias comparten o podrían compartir una misma estructura, desde recursos hardware, hasta sistema operativo, bases de datos o lógica de negocio.

En una arquitectura multitenant, los tenants comparten una misma estructura. Esto implica que consumen el servicio desde la misma plataforma tecnológica, incluyendo, por ejemplo, servidores, modelo de datos, o la capa de bases de datos. La arquitectura multitenant sigue el principio de arquitecturas software según el cual una única instancia de la aplicación se ejecuta en un servidor y da servicio a múltiples clientes. De esta manera, una arquitectura *multitenant* bien definida con una buena gestión de los recursos software y hardware, puede mejorar considerablemente los costes económicos y de mantenimiento de una aplicación [1][2][6]. En un entorno *multitenant* es el propio entorno el que debe proveer la capacidad de manejar todas los tenants de for-

ma transparente al usuario final. Dentro de la multitenencia encontramos varios niveles, desde el sistema operativo, al middleware o a la aplicación [17], o a los distintos niveles dentro del nivel de aplicación [7]:

- **Multitenencia a nivel de modelado de datos**: Todos los tenants comparten la misma BD y se establecen particiones lógicas que mantienen los datos separados entre sí. De esta manera cada tenant mantiene su independencia a nivel de datos con el resto [14].
- **Multitenencia a nivel de aplicación**: Los distintos tenants continúan compartiendo la BD y además comparten la misma instancia del producto software [8].
- **Multitenencia completa**: Todos los tenants comparten la misma BD y la misma instancia de software. Además de esto, cada uno posee su propia variante del producto, la cual se ajusta de forma personalizada a sus requisitos y necesidades. Este tipo de multitenencia se denomina completa por añadir las posibilidades de variabilidad al producto software. Esta multitenencia es la que se estudia en este artículo.

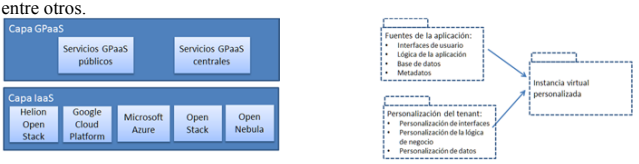
La personalización de los tenants está reconocida como uno de los grandes desafíos existentes [2][6], así como su adaptación al contexto. Lee y Choi [9] hacen hincapié en la dificultad existente en distribuir la lógica y los datos entre los distintos tenants, aun sin plantearse una configuración o adaptación en tiempo de ejecución. Muchos trabajos se han realizado para buscar soluciones para la personalización de los tenants: desde un estudio de la configuración de la multitenencia de la aplicación , en términos de seguridad, rendimiento, disponibilidad y gestión [6], o calidad de servicio [4], a estudios de mecanismos para definir la configuración de los tenants basados en conceptos de SPL [17] u Orientación a Aspectos [15][19]. Finalmente otro tipo de trabajos desarrollan casos de estudios donde destacan la necesidad de investigación en la configuración de la multitenencia y exploran posibilidades más específicas como las interfaces gráficas de usuario, la lógica del negocio o workflow, la selección de servicios o datos específicos [16]. Si bien es cierto, todas estas soluciones son dependientes del proveedor/plataforma en la nube sobre el que están implementadas y no consideran la adaptación en tiempo de ejecución.

3 ¿Por qué GPaaS?

Uno de los mayores problemas existentes hoy en día a la hora de plantear una solución en la nube a nivel de PaaS es la dependencia que existe con la plataforma en la nube sobre la que se desarrolla dicha solución. Por ello, es necesario crear soluciones a este problema, como por ejemplo adaptadores a las APIs de los distintos proveedores sobre los que se quiera desplegar dicha solución. Minsait (by Indra) con su solución en la nube GPaaS [12], ha dado un paso hacia adelante en ese sentido, proveyendo de una solución a la independencia de proveedor con su servidor de aplicaciones *multitenant* que puede ser implantado sobre las mayoría de las plataformas de la nube más conocidas (ver **Fig. 1**.a).

GPaaS [12] es una plataforma que se puede implantar tanto *on-premises*, como *off-premises* o como un servicio de una arquitectura en la nube híbrida. GPaaS es una solución *multitenant*, elástica y escalable gracias a sus capacidades de autoprovisionamiento y una base de datos basada en grafos. Pero su característica más destacable,

y por la que se ha elegido para este estudio, es su independencia del proveedor de IaaS, proporcionando un acceso homogéneo a una gran variedad de proveedores de la nube (ver capa IaaS, **Fig. 1**.a). El nivel GPaaS de la arquitectura se divide en dos bloques, los servicios públicos y los servicios centrales (ver capa GPaaS, **Fig. 1**.a). Los servicios públicos principalmente son un conjunto de servidores *multitenant* para distintos lenguajes de programación con el objetivo de hacer la plataforma, no sólo independiente de IaaS sino también independiente de lenguaje, entre ellos JavaEE, .NET, y lenguajes de scripting como Perl, PHP, Python y Ruby. Por otro lado, los servicios centrales constituyen los bloques básicos de la plataforma y los servicios nativos y relativos con la nube. Estos servicios son los que implementan las propiedades de multitenencia, autoprovisionamiento y escalabilidad elástica de la plataforma, entre otros.



a) Modelo de Arquitectura b) Multitenencia de una aplicación

Fig. 1. GPaaS

GPaaS proporciona una multitenencia completa a nivel de aplicación, gracias a la ejecución de aplicaciones a través de sus contenedores multitenant, llamados tenants. GPaaS gestiona las peticiones de los distintos tenants a través de una sola instancia lógica de aplicación, de esta manera los recursos asociados al entorno de ejecución se comparten entre todos los tenants. Entre sus características, está el hecho de que permite la gestión personalizada de cada tenant, la declaración explícita de la multitenencia y el despliegue automático de los tenants de cada aplicación, mediante la configuración individual de sus tenants, gracias a la posibilidad de desplegar en caliente las aplicaciones y añadir nuevas versiones sin afectar a las instancias en ejecución.

El servidor de aplicaciones guarda toda la información relativa a la interfaz de usuario, la lógica de la aplicación y la BD en carpetas físicas distintas dentro del repositorio compartido por los tenants. Los tenants son configurables a partir de metadatos. Estos metadatos permiten configurar la personalización de la aplicación para cada tenant, dando respuesta a uno de los desafíos existentes de soporte a la variabilidad en la nube [2][6] [9]. Una vez el tentant se ha configurado, GPaaS construye una aplicación con un unión virtual (*join*) entre la aplicación base y la personalización de cada tenant (ver **Fig. 1**.b). Esta unión virtual consiste en reutilizar la base de una aplicación, en el caso de Java, el war que contiene la lógica), para generar nuevos tenants, reescribiendo partes de la aplicación (.war) tales como recursos externos (ficheros de imágenes, html, css, etc.), archivos de configuración (.properties, ficheros .xml, etc.) o clases de la propia aplicación (.jar), para lograr una mayor capacidad de personalización de las aplicaciones. Esta unión (*join*) se describe en detalle en la sección 4.

4 Estudio del Soporte a la Variabilidad de SaaS *multitenant* en la Nube

La necesidad de soporte a la personalización y flexibilidad en la construcción de software en la nube es un hecho, y su adaptabilidad, todavía más. Hoy en día, se está trabajando en soluciones para dar soporte a estas propiedades a través de la multitenencia completa, si bien es cierto, adolecen de ser soluciones dependientes de proveedor. La solución en la nube GPaaS nos da solución al soporte a la multitencica completa e independiente de plataforma, es decir, una misma aplicación *multitenant* se puede ejecutar sobre distintas plataformas cloud. Como esta solución da un paso más allá que otras soluciones, se ha elegido GpaaS para realizar el estudio, con el objetivo departir de este punto y así encontrar qué necesidades quedan por cubrir y cómo podríamos enriquecer la flexibilidad en la configuración de la multitenencia, así como la adaptabilidad. El estudio se realizó siguiendo las líneas maestras de un caso de estudio, ya que están diseñados precisamente para investigar comportamientos, analizar teorías o tecnologías y obtener resultados en contextos reales entre otros. Por ello, se ha descrito siguiendo las guías de Runeson and Höst [13].

El estudio se ha realizado en el marco de un laboratorio iSmart Software Factory (iSSF) de la UPM. La iSSF es un marco para la educación e investigación en ingeniería del software en colaboración con empresa. En particular, este marco nos ha servido para desarrollar este caso de estudio UPM y Minsait (by Indra) distribuido entre las ciudades de Valencia (España) y Bucaramanga (Colombia). Por lo tanto, el desarrollo se ha llevado a cabo con 3 localizaciones geográficas diferentes.

El equipamiento y tecnología con la que cuenta la iSSF ha permitido desarrollar el trabajo adecuadamente, así como su monitorización y evaluación. La metodología de desarrollo ágil empleada ha sido SCRUM, mediante la definición de sprints de 15 días, al final de los cuales se organizaba una reunión de revisión del sprint (*review meeting & retrospective*). En la reunión se presentaban los progresos obtenidos y se establecían los objetivos, pautas y líneas del siguiente sprint, así como los problemas surgidos y como solventarlos y mejorar el equipo.

4.1 Diseño del caso de estudio

Esta sección describe el caso de estudio, en particular los objetivos que se persiguen, cómo se han recogido los resultados y se han analizado y los sujetos que han participado en el caso de estudio desarrollado [13].

Objetivos.

El objetivo principal del estudio se centra en explorar las opciones de variabilidad que nos ofrece GPaaS para definir líneas futuras de investigación, partiendo de la independencia de proveedor y la multitenencia completa que soporta dicha plataforma. Además de esto, se realizará un estudio sobre la adaptabilidad, es decir, el soporte a los cambios en tiempo de ejecución. En concreto, de los distintos servidores de aplicaciones y proveedores de la nube que proporciona la plataforma, las pruebas se han realizado sobre el servidor de aplicaciones Java EE (GAppServer) y se instalado sobre el proveedor de servicios de la nube (CSP) de Microsoft Azure.

Recolección de datos y Mecanismos de Evaluación.
La información de las distintas pruebas de concepto será almacenada utilizando los siguientes mecanismos:

- *RedMine*: Registro de Historias de Usuarios, Tiempos, Porcentajes de Desarrollo y Satisfacción de los resultados
- *Subversión*: Repositorio del código de los proyectos que representan las distintas pruebas de conceptos.
- Documentación *Review meetings*: documentos, diapositivas, vídeos y demostraciones de los resultados de cada sprint.
- Notas actas y correos: de las reuniones presenciales y por *Skype* establecidas.

Descripción del Caso de Estudio.

El caso de estudio para explorar la variabilidad es sencillo pero suficientemente representativo como para desarrollar las pruebas de concepto que se deseaban realizar con GPaaS. Éste consiste en explorar la variabilidad de un servicio de recomendación de restaurantes que es personalizable de acuerdo a las necesidades de distintos grupos/tipos de restaurantes. El servicio de la aplicación consiste en proporcionar las funcionalidades de votación y consulta de restaurantes para dos restaurantes distintos.

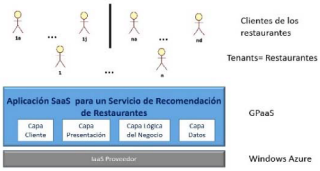


Fig. 2. Aplicación SaaS de GPaaS de Recomendación de Restaurantes (*adaptada de [17])*

En el marco de esta aplicación se va a explorar la multitenencia completa de la aplicación, haciendo un estudio de la variabilidad desde sus dos puntos de vista: variabilidad externa y variabilidad interna, así como el soporte de ambas variabilidades. Para ello se van a implementar tenants distintos para 2 restaurantes. Además de la variabilidad, se estudiarán los niveles de reconfiguración soportados en tiempo de ejecución de la aplicación base, adición de tenants y modificación de tenants.

Sujetos.

En total, han participado 8 personas en el caso de estudio (ver **¡Error! No se encuentra el origen de la referencia.**). Todos ellos han trabajado en conjunto para desarrollar la prueba de concepto presentada en este estudio, manteniendo frecuente contacto telefónico, presencial o por videoconferencia. Los analistas programadores han sido los encargados de producir el código de la prueba de concepto y testearlo bajo las directrices y supervisión del resto del equipo. El *scrum master* y el *product owner* han desarrollado la labor de gestión y liderazgo del proyecto, coordinando las distintas reuniones y supervisando el trabajo realizado en las distintas revisiones (*review meetings)* que han tenido lugar. Los arquitectos han diseñado la arquitectura de la prueba de concepto, verificándola para su posterior producción por parte de los

analistas programadores. Los instaladores se han encargado de instalar las infraestructuras en la nube y la GPaaS, y dar soporte a los problemas técnicos o de uso en base a los manuales GPaaS proporcionados. Los observadores han sido los encargados de supervisar los resultados de desarrollo e instalación, validando el trabajo realizado, los que han proporcionado la formación necesaria acerca de GPaaS vía presencial y en teleconferencia, y han solventado las preguntas acerca de la plataforma de desarrollo. En total, se han establecido 3 seminarios de formación, 2 reuniones presenciales entre los equipos de Madrid y Valencia, y 2 videoconferencias, así como múltiples correos de soporte técnico.

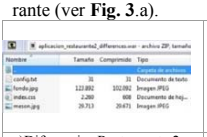
Tabla 1. Sujetos del Caso de Estudio

Sujeto	Rol	Experiencia en Cloud Computing	País / Ciudad	Organización
1	Analista / Programador	Bajo	Madrid - España	UPM
2	Analista / Programador	Medio	Madrid - España	UPM
3	Scrum Master / Arquitecto Software	Alto	Madrid - España	UPM
4	Product Owner / Arquitecto Software	Media	Madrid - España	UPM
5	Analista Cloud / Observador	Alto	Valencia - España	Minsait (by Indra)
6	Analista Cloud / Observador	Alto	Valencia - España	Minsait (by Indra)
7	Instalación / Soporte Técnico	Medio	Bucaramanga - Colombia	Minsait (by Indra)
8	Instalación / Soporte Técnico	Alto	Bucaramanga - Colombia	Minsait (by Indra)

4.2 Resultados de la Ejecución del Caso de Estudio

Prueba de Concepto de Variabilidad Externa: Variabilidad Capa de Presentación.

El servicio de recomendación de restaurantes se ha desarrollado con el lenguaje java usando J2EE, se ha desplegado en el servidor GAppServer mediante un paquete en formato war. La variabilidad externa se realizó mediante la definición de 2 tenants distintos para cada restaurante, de forma que la aplicación despliega una instancia capaz de responder a 2 tenants en ejecución. La declaración explícita de la existencia de diferencias en la aplicación implica especificar para cada tenant dos archivos *.tenant* que controlarán la ejecución de la aplicación y un archivo *.xml*. Por otro lado, también se define un *.war* de diferencias, que deberá seguir la misma estructura de directorios y ficheros de la aplicación original y contendrá aquellos objetos de la aplicación que desean ser reemplazados como pueden ser imágenes, estilos, etc. de la aplicación base. En particular, en este caso contiene los ficheros de estilo css, las imágenes de fondo y la imagen del restaurante 2 que varían con respecto al primer restaurante (ver **Fig. 3**.a).



a)Diferencias Restaurante 2

```
package es.indra.gmubila.gjserv2.config.interfaces

def Tenant restaurante1 = Tenant.config {
    name 'restaurante1'
}

Port.config(restaurante1)
{
    port 8103
    connector CONNECTOR_HTTP
}
```

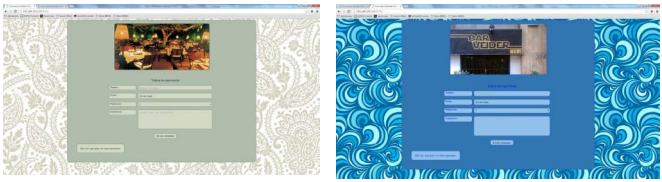
b) Tenant de Configuración de Despliegue Restaurante 1

```
appname=aplicacion
tenant_appname=restaurante1_aplicacion
merge_tenant_war=false
version=101
```

c) Tenant de configuración – metainformación Restaurante 1

Fig. 3. Configuración Multitenencia

Fig. 4. Definición de Tenants de la Aplicación de Servicio de Recomendación de Restaurantes



Restaurante 1 Restaurante 2

Fig. 5. Ejecución de los tenants de los restaurantes 1 y 2 con variabilidad en la presentación

De los dos ficheros *.tenant* que hay que definir para definir un tenant, en uno de ellos se especifica el puerto en el que se despliega y el nombre del tenant. Esta información es la que permite desplegar la aplicación, por lo que si no existe este archivo la aplicación no será desplegada (ver ejemplo del tenant del restaurante 1 **Fig. 3**. b). En el otro fichero *.tenant* se ha de especificar el nombre de la aplicación, el nombre del tenant seguido de un guion bajo y el nombre de la aplicación, además de indicar si tiene o no un fichero de diferencias mediante un valor booleano y el número de versión de la aplicación. En la **Fig. 3**.c se especifica este archivo *.tenant* para el restaurante 1.Como su información de la capa de presentación es la misma que la de la aplicación base, el valor booleano para el fichero de diferencias es false, sin embargo, en el caso del archivo *tenant* del restaurante 2, es true. Una vez acabadas las configuraciones de los tenants nuestra variabilidad está conformada por los archivos que muestra la **Fig. 4** y el resultado de ejecución que muestra la **Fig. 5**.

Prueba de Concepto de Variabilidad Interna: Variabilidad Capa de Datos.

En este caso, se mantiene una única capa de presentación y de lógica de negocio, pero se va a cambiar los esquemas de la base de datos. Para ello, además de los archivos configurados en la prueba de concepto anterior, se deberá configurar el archivo de recursos del tenant, para indicar cuál es el esquema en cada caso (ver **Fig. 6**.a). El fichero *.xml* de recursos, declara la conexión a la base de datos, sin necesidad de manipularla desde la propia aplicación, que se encargará del manejo de los datos y no de la conexión. Además, se debe indicar la ruta donde encontrar los datos que se utilizarán en la aplicación. Para esta prueba de concepto se han creado dos esquemas (*schemas)* en la base de datos PostgreSQL de la aplicación (**Fig. 6**. b). Dichos esque-

mas implementan el mismo modelo de datos con el objetivo de proveer de la misma funcionalidad pero aislando los datos entre instancias.

a) Archivo de recursos del tenant restaurante 1

b) Esquemas de la BD para el Restaurante 1 y el restaurante 2

Fig. 6. Configuración de recursos y esquemas de la BD

Prueba de Concepto de Variabilidad Interna y Externa

Esta prueba de concepto permitió aplicar ambas variabilidades a las distintas instancias lógicas de una misma aplicación por lo que el cliente aprecia aplicaciones distintas en las que una lógica interna común dirige la aplicación.

En primer lugar, dada la configuración básica de los dos tenants, los usuarios de los dos tenants obtienen presentaciones y datos diferentes a pesar de trabajar sobre la misma lógica de la aplicación (ver **Fig. 7**).

Restaurante 1 Restaurante 2

Fig. 7. Ejecución de variabilidad externa e interna

Otra configuración que se realizó para que coexistiesen ambas variabilidades fue incluir una tabla que guardaba los miembros del *staff* en uno sólo de los tenants, concretamente en el restaurante 2. A través del código de la aplicación se identificaba si el esquema de la BD tenía una tabla *staff* o no, y en caso de existir dicha tabla, automáticamente se modifica la IGU del tenant 2, para así mostrar una opción adicional en el formulario de creación de comentarios. De esta manera es posible combinar ambas

variabilidades, alterando la interfaz gráfica de usuario y la lógica de negocio de manera independiente en cada tenant. En caso de no existir dicha tabla, la GUI mostrará los campos por defecto.

Prueba de Concepto de Reconfiguración Dinámica

Una de las características de GPaaS es la capacidad de desplegar aplicaciones en caliente sobre la plataforma. Por ello dentro de las pruebas de concepto anteriores se realizaron pruebas que permitieran evaluar el grado de adaptabilidad de las aplicaciones *multitenant* desarrolladas con GPaaS.

Durante el despliegue de la aplicación concerniente a la prueba de concepto de la variabilidad externa se hicieron cambios en caliente sobre la lógica de la aplicación y los cambios surtieron efecto. Asimismo, la inclusión del segundo tenant, el segundo restaurante, se hizo en caliente y se pudo lanzar el servicio sin que el servicio restaurante 1 fuera afectado. Finalmente, hicimos una prueba con la opción que ofrece definir tenants maestros y esclavos y los cambios de uno se trasladaban a los otros. Estas pruebas fueron satisfactorias. Sin embargo, en el caso de intentar hacer cambios en la personalización de los propios tenants, tanto en las imágenes y css mostrados en la variabilidad externa como en los recursos de los esquemas de la BBDD de la variabilidad interna, no surtieron efecto en caliente. Por lo tanto, en caso de querer modificar la configuración de un tenant ya existente, es necesario reiniciar GAppServer para que los cambios surtan efecto.Dicho comportamiento ha sido debidamente reportado y desde Minsait (by Indra) ya se está trabajando para proveer dicha capacidad.

4.3 Conclusiones del estudio

Este caso de estudio nos ha ayudado a realizar pruebas de concepto de desarrollo de aplicaciones *multitenant* sobre Micosoft Azure, sin tener conocimientos de este proveedor de servicios, simplemente utilizando un desarrollo Java sencillo sobre un entorno como Eclipse al uso, configurando 3 archivos para definir la variabilidad requerida para cada tenant. Por tanto, la experiencia en cuanto a abstracción de plataforma ha resultado satisfactoria. A este hecho hay que añadir que el estudio nos ha mostrado que se da soporte a una multitenencia explícita y completa a nivel de aplicación tal cual se define en la literatura (datos, lógica y completa). Esto supone un avance notable en el ámbito del desarrollo puesto que una vez creada una aplicación, es posible modificarla a partir de las distintas variabilidades permitidas con el objetivo de crear multitud de aplicaciones diferentes. Este hecho elimina la necesidad de desarrollar aplicaciones desde cero, seleccionando aquellas facetas o características a diferenciar de la aplicación original. De esta manera es posible crear variantes de una misma aplicación siempre que se mantenga un núcleo común. Además de esto, el soporte al despliegue en caliente de nuevos tenants y modificaciones en la aplicación base están soportados. Llegados a estas conclusiones, podemos asumir como hallazgo que este es el punto de partida de investigaciones futuras y que de este caso de estudio se han determinado dos acciones principales que a día de hoy la comunidad científica debe cubrir:

- Con respecto a la variabilidad, se ha detectado que únicamente se soporta un nivel de variabilidad, cuando se podrían establecer múltiples niveles de variabilidad reutilizables, pudiendo hacer compartir tenants a un conjunto de subtenants, aumentando el nivel de reutilización. Por ejemplo, si dentro de un mismo producto (pongamos por caso agencias de viajes), estas compartieran una serie de elementos (aplicación base), se diferenciaran a nivel autonómico (tenants) pero cada oficina tuviera una gestión específica podrían emplearse subtenants para especificar dicha gestión de forma explícita. Lo mismo en el área de restauración se podría aplicar a las franquicias y restaurantes específicos.
- Con respecto a la adaptabilidad, si queremos una sensibilidad al contexto y la adaptación completa en tiempo de ejecución, necesitamos mecanismo que permita realizar el despliegue en caliente no sólo de nuevos tenants, sino también la modificación de los tenants existentes.

5 Conclusiones y Trabajos futuros

En este trabajo, se ha presentado un estudio del soporte actual a la variabilidad en el desarrollo de software en la nube y se ha dado a conocer una solución como GPaaS, que da un paso más allá, ofreciendo la independencia de proveedor. Además, el trabajo presenta un caso de estudio en el que se hace un análisis detallado de la implementación de la variabilidad a través de la multitenencia, así como de la configuración de dinámica de dicha variabilidad en la nube. Este trabajo ha permitido identificar las necesidades y trabajos futuros que la comunidad de arquitecturas debe resolver, haciéndolos patentes en este artículo.

Los trabajos futuros, precisamente, se fundamentan en los resultados obtenidos del caso de estudio: dar soporte a la reconfiguración dinámica de tenants y diseñar mecanismos de definición de subtenants y mecanismos de configuración de aplicaciones en base a tenats y subtenants.

6 Referencias

[1] Bezemer, C., Zaidman, A.: Multi-tenant SaaS applications: maintenance dream or nightmare? Int. Workshop on Principles of Software Evolution (IWPSE), pp. 88–92. ACM, NY 2010.

[2] Bezemer, C., Zaidman, A., 2010. Challenges of reengi-neering into multitenant SaaS applications. Tech-nical Report of Delft Uni. of Technology, TUD-SERG-2010-012, 2010

[3] Bosch, J.: Software architecture: The next step. In: Software Architecture Lecture Notes in Computer Science Volume 3047, pp. 194-199 (2004)

[4] Fehling, C., Leymann, F., Mietzner, R., 2010: A framework for optimized distribution of tenants in cloud applications. IEEE 3rd Int. Conference on, Cloud Computing (CLOUD), 2010, pp. 252-259.

[5] Gold, N., Mohan, A., Knight, C., Munro, M.: Understanding service-oriented software. IEEE Software 21(2), 71–77 (2005)

[6] Guo, C., Sun, W., Huang, Y., Wang, Z., Gao, B., 2007: A framework for native multitenancy application devel-opment and management. CEC/EEE 2007: Int. Conf. On Enter-

prise Computing, E-Commerce Tech-nology and Int. Conf. On Enterprise Computing, E-Commerce and E-Services, pp. 551-558 (2007)

[7] Kabbedijk, Jaap and Jansen, Slinger "Variability in Multi-tenant Environments: Architectural Design Patterns from Industry". Advances in Conceptual Modeling. Recent Developments and New Directions. Springer Berlin Heidelberg, 2011, pp. 151--160

[8] Kwok, T., Nguyen, T., Lam, L.: A software as a service with multi-tenancy support for an electronic contract management application. IEEE International Conference on Services Computing, SCC 2008, vol. 2, pp. 179–186. IEEE, Los Alamitos (2008)

[9] Lee, W., Choi, M., 2012. A multi-tenant web application framework for SaaS. In 2012 IEEE 5th Int. Conf. on Cloud Computing (CLOUD), 2012, pp. 970–971.

[10] Meyer, N.H. and Lehnerd A. P., The power of product platforms: building value and cost for leadership, free press, New York, 1997.

[11] Pohl K, Böckle G, Linden F (2005) Software product line engineering: foundations, principles and techniques. Springer, Germany

[12] Rios A., Paraire. J, Valencia A.M., GPaaS White Paper, Cloud Expert Centre, Minsait, Indra, 2015. http://www.gnubila.com/

[13] Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empirical Softw Eng 14:131–164

[14] Schiller, O., Schiller, B., Brodt, A., Mitschang, B.: Native support of multi-tenancy in RDBMS for software as a service. In: Proceedings of the 14th International Conference on Extending Database Technology, pp. 117–128. ACM, New York (2011)

[15] Shahin, A., Samir, A., Khamis, A., 2013. An Aspect-Oriented Approach for SaaS Application Customization. 48th Conf. on Statistics, Computer Science and Operations Research, Cairo University, Egypt, 2013.

[16] Tsai, W., Shao, Q., Li, W., 2010. OIC: Ontology-based intelligent customization framework for SaaS. In IEEE Int. Conf. on Service-Oriented Computing and Applications (SOCA), 2010, pp. 1–8

[17] Walraven, S., Truyen, E., W. Joosen, W., 2011. A middle-ware layer for flexible and cost-efficient multi-tenant applications. Proc. on Middleware, 2011 (LNCS 7049), pp. 370-389.

[18] Wang, L., Ranjan, R., Chen, J., & Benatallah, B. (Eds.). (2011). Cloud computing: methodology, systems, and applications. CRC Press.

[19] Wang, H., Zheng, Z., 2010. Software Architecture Driven Configurability of Multi-tenant Saas Applications. LNCS Vol. 6318, 2010 pp. 418-424.

Evolución arquitectónica de servicios basada en modelos CVL con cardinalidad

José Miguel Horcas, Mónica Pinto, and Lidia Fuentes

Universidad de Málaga, Andalucía Tech, Spain
{horcas,pinto,lff}@lcc.uma.es, http://caosd.lcc.uma.es/

Resumen La computación en la nube se está convirtiendo en un mecanismo predominante para desplegar fácilmente aplicaciones con requisitos especiales, tales como el almacenamiento masivo compartido, o el equilibrado de carga. Esta funcionalidad se proporciona normalmente como servicios por las plataformas en la nube. Un desarrollador puede mejorar tanto el despliegue de sus aplicaciones como la productividad siguiendo un enfoque *multi-tenancy*, donde diferentes variantes de la misma aplicación pueden adaptarse rápidamente a las necesidades de cada usuario (*tenant*). Sin embargo, gestionar la variabilidad inherente a las aplicaciones multi-tenant, con cientos de usuarios y miles de configuraciones arquitectónicas diferentes, puede llegar a ser una tarea intratable de abordar manualmente. En este artículo, se propone un enfoque de línea de producto software en el cual: (1) usamos modelos de variabilidad con cardinalidad para modelar cada tenant como una característica clonable, (2) automatizamos el proceso de evolución de las arquitecturas de aplicaciones multi-tenant, y (3) demostramos que la implementación de los procesos de evolución es correcta y eficiente para un número elevado de tenants en un tiempo razonable.

Keywords: Cardinalidad, Evolución, Línea de Producto Arquitectónica, Variabilidad, CVL

1. Introducción

La computación en la nube se está convirtiendo en el principal mecanismo para desplegar fácilmente aplicaciones con requisitos especiales, tales como el almacenamiento masivo compartido, escalado automático, o el equilibrado de carga [2]. Los desarrolladores pueden integrar los servicios ofrecidos por las plataformas en la nube (e.g., Microsoft Azure, Amazon Web Services) como parte de la arquitectura de sus aplicaciones, disminuyendo así el tiempo de desarrollo.

Diferentes versiones de la misma aplicación pueden ser desarrolladas siguiendo un enfoque *multitenancy* [12], donde cada variante de una aplicación puede ser personalizada según las necesidades de cada usuario (*tenant*). Sin embargo, gestionar la variabilidad inherente en las aplicaciones multi-tenant, donde es necesario mantener diferentes configuraciones de la arquitectura software para cada tenant, no es una tarea sencilla. Las Líneas de Producto Software (SPL, del inglés *Software Product Line*) [15] constituyen un enfoque ampliamente usado para especificar la variabilidad en general, y específicamente en arquitecturas orientadas a servicios [5]. Es aquí dónde las aplicaciones multi-tenant plantean

un nuevo reto: representar de forma explícita y gestionar la existencia de configuraciones simultáneas de los mismos servicios, uno para cada tenant [7].

Además, se debe tener en cuenta la gestión de la evolución de este tipo de aplicaciones. Los proveedores de las plataformas en la nube están continuamente evolucionando y actualizando sus tecnologías con el fin de ser competitivos en el mercado. Por otra parte, la funcionalidad específica de la aplicación puede evolucionar para tener en cuenta nuevas características solicitadas por los usuarios. En ambos casos, la arquitectura software de la aplicación en la nube debe ser adaptada para añadir nuevos componentes software o eliminar y reconfigurar los existentes. Sin embargo, gestionar la evolución de una aplicación multi-tenant con cientos de usuarios y miles de configuraciones posibles puede llegar a ser una tarea inabordable manualmente. Aunque existen SPLs en el contexto de las aplicaciones multi-tenant [5,9,14,19], la mayoría presenta dos principales limitaciones: (i) no tienen en cuenta la automatización de la evolución a nivel de la arquitectura software; y (ii) instancian la SPL de forma individual para cada tenant. Esto dificulta la realización de los cambios de forma automática y consistente, y aumenta la complejidad de cambiar simultáneamente las configuraciones arquitectónicas existentes para cada tenant.

Los principales objetivos de este artículo son: (1) identificar los cambios necesarios en las aplicaciones cuando los requisitos, tanto de la propia aplicación como de los servicios proporcionados por la plataforma en la nube, cambian; y (2) obtener de manera simultánea, para cada tenant, una arquitectura software evolucionada que sea válida y consistente con la configuración actualmente desplegada en ese tenant. Para lograr estos objetivos se propone una línea de productos arquitectónica (PLA) [4] en la que: (i) modelamos la configuración de cada tenant como una característica clonable usando modelos de variabilidad con cardinalidad en CVL [10]; (ii) automatizamos el proceso de evolución de la arquitectura multi-tenant definiendo tres algoritmos que propagan automáticamente los cambios necesarios en la configuración arquitectónica desplegada en cada tenant; y (iii) demostramos que los algoritmos son correctos y eficientes para un número elevado de tenants. Ilustramos nuestra propuesta con una aplicación en el dominio del software médico.

El artículo se organiza de la siguiente manera. La Sección 2 describe los retos principales de nuestra propuesta a través de un caso de estudio. La Sección 3 explica como modelar la variabilidad de las aplicaciones multi-tenant con CVL. La Sección 4 y 5 detallan nuestro proceso de evolución y los algoritmos propuestos. La Sección 6 evalúa nuestra propuesta. Finalmente, la Sección 7 discute el trabajo relacionado y la Sección 8 las conclusiones y el trabajo futuro.

2. Motivación y caso de estudio

Nuestro caso de estudio es una aplicación para la gestión y administración de servicios médicos en hospitales. Con el fin de ahorrar en costes de implementación y mantenimiento, se decide desarrollar la aplicación usando una plataforma en la nube (e.g., Microsoft Azure) [18]. El objetivo es vender la aplicación médica a diferentes clientes (hospitales), por lo que para tener configuraciones diferentes para cada hospital, la aplicación seguirá un enfoque multi-tenant, considerando

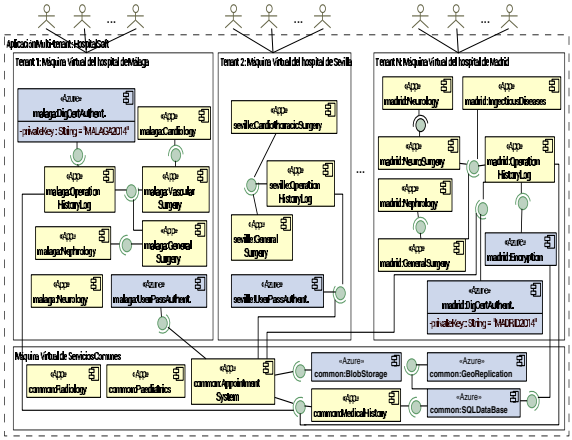


Figura 1. Arquitectura software de una aplicación multi-tenant en Microsoft Azure.

cada hospital como un tenant. Además, se plantea utilizar varios de los servicios proporcionados por la plataforma Azure, como la persistencia y la geo-replicación de datos. Por lo tanto, la aplicación multi-tenant tendrá un conjunto de servicios comunes disponibles desde la misma máquina virtual y compartidos por todos los usuarios, y también un conjunto de servicios específicos para cada usuario disponibles a través de máquinas virtuales específicas para cada usuario. La Figura 1 muestra una instancia de la arquitectura de la aplicación médica instanciada y configurada para tres usuarios: los hospitales de Málaga, Sevilla y Madrid.

Primero, se proporciona un conjunto de servicios que son comunes para todos los tenants desde la misma máquina virtual (`Máquina Virtual de Servicios Compunes`). Algunos de ellos, estereotipados como **«App»**, son específicos de la aplicación médica, como el sistema de citas (componente `Appointment System`) y el historial médico de los pacientes (`Medical History`). Otros, estereotipados como **«Azure»**, son servicios ofrecidos por la plataforma de Microsoft, como el servicio de persistencia para almacenar los datos clínicos (`SQL DataBase`) y las citas de los pacientes (`BlobStorage`), o la posibilidad de crear múltiples copias de la información en diferentes centros de datos (`GeoReplication`). Segundo, además de estos servicios comunes, la aplicación proporciona para cada tenant un conjunto de servicios específicos configurados acorde a sus diferentes necesidades. Por ejemplo, el hospital de Málaga es el único que realiza cirugías vasculares, por lo que el componente `VascularSurgery` está incluido únicamente en este tenant. Análogamente, los componentes `Nephrology` y `Neurology` están instanciados para el hospital de Málaga y de Madrid, pero no para el hospital de Sevilla. Finalmente, no sólo los componentes específicos de la aplicación varían entre los diferentes tenants, algunos servicios de la plataforma en la nube también pue-

den ser configurados por cada tenant. Por ejemplo, el método de autenticación es diferente para cada hospital: el hospital de Málaga usa un certificado digital para autenticar al personal médico en el sistema (componente `DigCertAuthent`) y nombre de usuario y contraseña para autenticar a los pacientes en el servicio de cita médica en línea (`UserPassAuthent`); mientras que el hospital de Sevilla usa autenticación mediante el nombre de usuario y contraseña para todos indistintamente, y el hospital de Madrid usa certificado digital para todos.

En las aplicaciones multi-tenant hay características (implementadas como componentes) que son requeridas por todos los tenants y otras que son variables y configurables. Esto significa que normalmente sólo un subconjunto de las características variables de la aplicación son desplegadas en cada tenant. Teniendo en cuenta que se deben generar y mantener cientos de configuraciones diferentes de la aplicación, lo que implica la gestión de miles de componentes, un reto importante es *proporcionar mecanismos para representar y gestionar directamente la variablidad de las aplicaciones multi-tenant.*

Pero una vez desplegadas, las aplicaciones multi-tenant tienen que ser adaptadas a mejoras tecnológicas y a cambios en las necesidades de los usuarios. Por ejemplo, nuevos métodos de autenticación (e.g., autenticación biométrica, autenticación usando redes sociales) o persistencia (e.g., fragmentación de base de datos, grupos de afinidad) aparecen con frecuencia. Si se quiere proporcionar estos nuevos servicios a los clientes, éstos se deben incorporar a los tenants que lo requieran. También se podría cambiar el proveedor de la plataforma en la nube y migrar la aplicación a una nueva (ej: Amazon Web Services). En este caso, la parte de la arquitectura de la aplicación que depende de los servicios de la plataforma tiene que adaptarse a los servicios que ofrece la nueva plataforma en la nube. Por último, durante la vida de la aplicación, los clientes pueden exigir nuevas funcionalidades (e.g., un nuevo módulo para la gestión de transplantes o cambios en los métodos de autenticación para identificar a los pacientes).

Considerando los cambios que es necesario realizar en la arquitectura software de la aplicación multi-tenant, todas las situaciones mencionadas anteriormente se pueden representar con tres escenarios diferentes de evolución: (1) un nuevo componente necesita ser incorporado en la arquitectura software, ya sea proporcionado por la plataforma en la nube (Azure) o implementado por el desarrollador de la aplicación; (2) un componente existente necesita ser eliminado de la arquitectura software; y (3) un componente existente necesita ser reconfigurado con nuevos parámetros. Sin embargo, la evolución de una aplicación multi-tenant implica también tener que calcular y realizar estos cambios para miles de componentes que se ejecutan en cientos de tenants, convirtiendo el proceso de evolución en una tarea intratable de abordar manualmente. Por lo tanto, otros retos importantes de la evolución en aplicaciones multi-tenant son *el cálculo automático de los cambios que se deben realizar en cada tenant* y *la propagación automática de estos cambios para todos los tenant a nivel arquitectónico.* Por otra parte, este proceso de evolución automático sólo será útil si es correcto y eficiente para un gran número de tenants, por lo que necesitamos *demostrar la eficiencia y la corrección del proceso de evolución.*

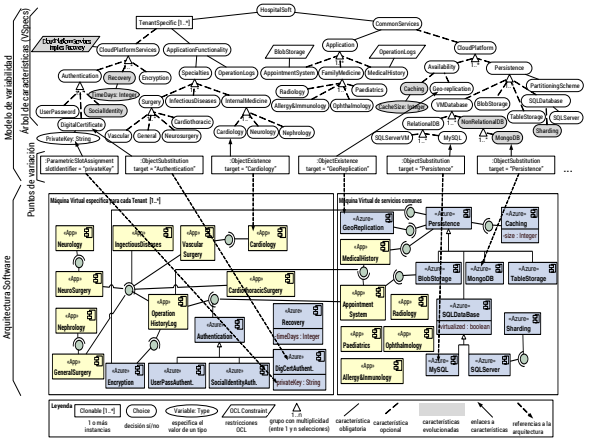


Figura 2. Modelo de variabilidad en CVL y arquitectura software.

3. Gestión de la variabilidad con CVL

En esta sección explicamos cómo nuestra propuesta usa CVL (*Common Variability Language*) [10] para gestionar la variabilidad de la arquitectura software de todos los tenant en una aplicación basada en la nube. CVL es un lenguaje independiente del dominio para especificar y resolver la variabilidad en modelos basados en MOF.[1] Concretamente, como muestra la Figura 2 para nuestro caso de estudio, modelamos explícitamente la variabilidad de las características que son específicas de cada tenant (i.e., sub-árbol `TenantSpecific[1..*]`) y la funcionalidad común que comparten todos los tenants (i.e., sub-árbol `CommonServices`).

El **modelo de variabilidad** CVL está formado por dos partes. La primera es una parte abstracta que modela las características opcionales y obligatorias (*VSpecs* en CVL, de *Variability Specifications*) y las restricciones entre ellas (*cross-tree constraints*). Esta parte abstracta se define mediante un árbol como el mostrado en la parte superior de la Figura 2 y especifica la funcionalidad de la aplicación y los servicios ofrecidos por la plataforma en la nube. La segunda parte del modelo de variabilidad son los puntos de variación (*variation points*), que aparecen en la parte central de la Figura 2. Cada punto de variación está asociado a una característica del árbol y tiene una o más referencias a elementos de la arquitectura software. Estos puntos de variación representan modificaciones específicas a realizar en la arquitectura (i.e., transformaciones modelo a modelo, M2M) cuando una característica ha sido seleccionada en una configuración concreta del modelo de variabilidad. Cuando hablamos de definir una **configuración del modelo de variabilidad** nos referimos a seleccionar un conjunto

[1] `http://www.omg.org/mof/`

de características en el árbol que cumplan el conjunto de restricciones. Luego, el motor de ejecución de CVL es el encargado de ejecutar las transformaciones M2M asociadas a cada punto de variación según las características seleccionadas. En CVL una configuración del modelo de variabilidad recibe el nombre de **modelo de resolución** (*resolution model).*

Para dar soporte a diferentes configuraciones para cada tenant, nuestra propuesta define la funcionalidad específica de los tenant bajo una característica *clonable* (`TenantSpecific[1..*]` en la Figura 2). La característica clonable tiene una cardinalidad [1..*] que indica que esta característica puede ser instanciada una o más veces y todas sus sub-características pueden ser configuradas de forma diferente para cada instancia. En nuestro ejemplo, la cardinalidad representa el número de tenants, y cada instancia de `TenantSpecific[1..*]` define la configuración para ese tenant específico (por ejemplo para el hospital de Sevilla).

Seleccionando las características apropiadas bajo la característica clonable `TenantSpecific[1..*]`, y ejecutando CVL, nuestra propuesta genera una configuración de la arquitectura como la mostrada en la Figura 1, donde cada tenant está configurado de acuerdo a sus necesidades.

4. Gestión de la evolución con CVL

Una vez que se ha generado y desplegado una configuración arquitectónica adaptada a los requisitos de cada tenant, la aplicación multi-tenant es susceptible de evolucionar debido a mejoras tecnológicas y/o a cambios en las necesidades de los clientes. Volviendo a nuestra aplicación médica, supongamos que Microsoft incorpora nuevas funcionalidades a su plataforma: un servicio de recuperación de datos, un método de autenticación basado en Facebook y un mecanismo de persistencia. Supongamos también que el proveedor de la aplicación médica quiere proporcionar estos nuevos servicios a sus tenants (i.e., a sus hospitales).

El primer paso en el proceso de evolución es adaptar el modelo de variabilidad con nuevas características (aparecen en color gris en la Figura 2), y la arquitectura de la aplicación con nuevos elementos arquitectónicos (la parte inferior de la Figura 2 muestra la arquitectura ya evolucionada). Por ejemplo, las características `Recovery`, `SocialIdentity`, `Caching`, `MongoDB` y `Sharding` han sido incorporadas en el modelo de variabilidad con el fin de añadir el nuevo servicio de recuperación, el nuevo método de autenticación, y el nuevo mecanismo de persistencia (una nueva base de datos no relacional y un nuevo mecanismo de partición de base de datos). Así mismo, la arquitectura ha sido adaptada con los nuevos componentes `Recovery`, `SocialIdentityAuth` y `MongoDB`, entre otros.

El segundo paso en el proceso de evolución es calcular de manera automática y consistente los cambios que son necesarios realizar en todos los tenants que están actualmente desplegados (Figura 1). El objetivo es que esas configuraciones ya desplegadas satisfagan el nuevo modelo de variabilidad y los nuevos requisitos de la aplicación. Para automatizar esta tarea nuestra propuesta divide este segundo paso en dos partes: (i) modificar la configuración actual de todos los tenants, generando una nueva configuración evolucionada a partir del modelo de variabilidad previamente evolucionado (algoritmo *Evolve Configuration*); y (ii)

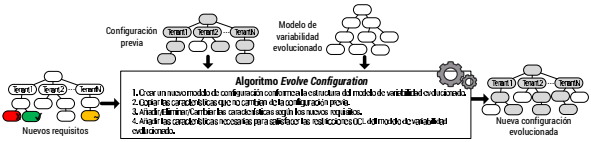


Figura 3. Algoritmo para evolucionar una configuración.

calcular las diferencias entre la configuración evolucionada y la configuración actualmente desplegada (algoritmo *Difference Configuration*). Recordamos que por configuración nos referimos a una instancia concreta del árbol de características.

El tercer paso es el más complicado en el proceso de evolución y consiste en propagar automáticamente los cambios previamente calculados a la arquitectura software actualmente desplegada. Para hacer esto, definimos un tercer algoritmo (*Create Weaving Model*) que genera un nuevo modelo en CVL que especifica cómo propagar a la arquitectura software las modificaciones definidas previamente a nivel de características (nos referimos a este modelo como *modelo de composición*). Este algoritmo es una de las principales contribuciones del artículo, debido a que las propuestas existentes que gestionan la evolución con SPLs (como en [1,8]) sólo abordan el problema de la evolución a nivel abstracto de características, y requieren modificar manualmente la arquitectura software para reflejar los cambios calculados — e.g., modificar el fichero de configuración de cada tenant, o definir manualmente un mapeo entre el modelo de configuración a nivel de características y la arquitectura evolucionada para cada tenant. Finalmente, CVL es ejecutado con el modelo de composición como entrada con el fin de obtener la arquitectura evolucionada con los cambios para cada tenant.[2]

4.1. Evolución del modelo de configuración

El algoritmo para evolucionar la configuración de una aplicación multi-tenant (*Evolve Configuration* en la Figura 3) recibe como entrada el modelo de la configuración actualmente desplegada (que será evolucionado), el modelo de variabilidad evolucionado (actualizado previamente por el proveedor de la aplicación), y la lista de las nuevas características requeridas por los clientes; y genera el modelo de la configuración evolucionada. El modelo generado representa, a nivel del árbol de características, una nueva configuración válida de la aplicación multi-tenant con todas las configuraciones de los tenants evolucionados.

La Figura 4 muestra una vista parcial del árbol de características donde, por limitación de espacio, se ha representado en el mismo árbol las tres entradas del algoritmo. El algoritmo genera un nuevo modelo donde, en primer lugar, se copian aquellas características que no cambian de la configuración previa (características en color blanco). A continuación, se añaden las nuevas características seleccionadas (marcadas con ✓), se omiten aquellas características que ya no son requeridas (marcadas con ×) y se añaden las características cuyos valores han cambiado (marcadas con ∼). Finalmente, se añaden aquellas características

[2] La formalización de los algoritmos y su definición completa está disponible en `http://caosd.lcc.uma.es/papers/evolutionAlgorithms.pdf`.

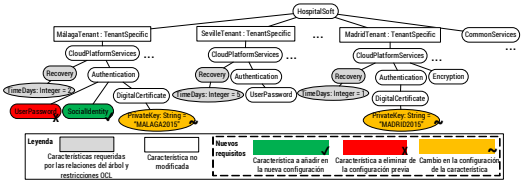


Figura 4. Información gestionada por el algoritmo *Evolve Configuration*.

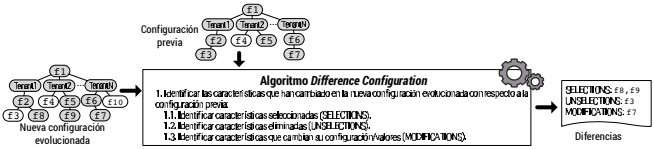


Figura 5. Algoritmo para calcular la diferencia entre dos configuraciones.

requeridas por las nuevas restricciones definidas en el modelo de variabilidad evolucionado (en gris). En nuestro ejemplo, para el tenant Málaga, la característica `UserPassword` es eliminada, mientras que `SocialIdentity` es añadida como nuevo requisito. Además, el parámetro `PrivateKey` del certificado digital es actualizado a un nuevo valor tanto para el tenant Málaga (‘‘`MALAGA2015`’’) como para el tenant Madrid (‘‘`MADRID2015`’’). También la característica `Recovery` y su parámetro `TimeDays`, que especifica el intervalo de copia de seguridad, son añadidos en todos los tenants debido a la nueva restricción (`CloudPlatformServices implies Recovery`) en el modelo de variabilidad evolucionado.

Una vez que el modelo de resolución evolucionado ha sido generado, el siguiente algoritmo calcula la diferencia entre la nueva y la anterior configuración.

4.2. Cálculo de las diferencias entre configuraciones

El algoritmo *Difference Configuration* recibe como entrada dos configuraciones (i.e., la configuración previa y la configuración nueva obtenida con el algoritmo *Evolve Configuration*) y calcula la diferencia entre ellas (Figura 5). Las diferencias están determinadas por: (1) las nuevas características seleccionadas en la nueva configuración que no estaban presentes en la anterior (`SELECTIONS`); (2) las características de la configuración anterior que han sido eliminadas en la nueva (`UNSELECTIONS`); y (3) las características que permanecen en la nueva configuración pero cambian sus valores con respecto a la anterior (`MODIFICATIONS`).

5. Propagación de los cambios a la arquitectura

En esta sección definimos el tercer algoritmo de nuestro proceso de evolución, que genera un modelo arquitectónico en CVL para propagar los cambios a la arquitectura desplegada. En primer lugar, con el fin de definir el algoritmo de forma precisa, es necesario formalizar los diferentes modelos de CVL — es decir, el modelo de variabilidad y los modelos de resolución. La formalización de CVL se encuentra parcialmente publicada en [6], pero sólo formaliza la parte abstracta (es decir, el árbol de características) del modelo de variabilidad, y no

los puntos de variación ni los modelos de resolución. Como parte de este trabajo, completamos la especificación definida en [6] para formalizar completamente los modelos de variabilidad y las configuraciones en CVL.

5.1. Formalización de los puntos de variación en CVL

Los puntos de variación (VPs, de *variation points*) definen los elementos del modelo arquitectónico que son variables y pueden ser modificados. Estos también especifican cómo esos elementos variables se modifican mediante transformaciones de modelo (e.g., en ATL [11]). La semántica de estas transformaciones es específica de cada tipo de punto de variación. Por ejemplo, algunos puntos de variación soportados por CVL son la existencia o no de elementos en la arquitectura (`ObjectExistence`), la existencia de relaciones entre los elementos (`LinkExistence`), o la asignación de un valor a una variable (`ParametricSlotAssignment`), entre otros [6].Un tipo importante de punto de variación es `Opaque Variation Point` (`OVP`) que permite definir nuevos puntos de variación personalizados y, por lo tanto, nuevas transformaciones de modelo que no están predefinidas en CVL. Durante la ejecución de CVL, el motor CVL delega su control en un motor de transformaciones modelo a modelo (M2M) encargado de ejecutar las transformaciones definidas por los puntos de variación.

Para representar los puntos de variación, definimos una tupla: $variationPoints = (VP, type, ovptype, semantic, binding, MOFRefs)$, cuyos elementos son:

VP. Conjunto finito, no vacío, de identificadores (nombres únicos) de los puntos de variación.
$type : VP \rightarrow VPType$. Función que dado un punto de variación devuelve su tipo de la taxonomía de puntos de variación disponible en CVL.
$ovpType : VP \rightarrow OVPType$. Función parcial que dado un OVP, devuelve el tipo de ese OVP.
$semantic : OVPType \rightarrow SemanticSpec$. Función que devuelve la semántica de un tipo de OVP. Esto incluye tanto la transformación de modelo a ejecutar por el motor M2M de CVL, como el lenguaje de transformación (e.g., ATL) usado por la transformación.
$binding : VP \rightarrow VSPEC$. Devuelve la característica del árbol asociada al punto de variación.
$refs : VP \rightarrow P(MOFRef)$. Función que devuelve las referencias a elementos de la arquitectura que están enlazadas con el punto de variación. A esos elementos se le aplicarán las transformaciones.

5.2. Formalización de los modelos de resolución en CVL

Dado un modelo de variabilidad V, un modelo de resolución R para V es una colección de características seleccionadas o resueltas ($VSPEC_{res}$) del modelo de variabilidad V. Estas características pueden ser de tres tipos según el tipo de resolución que requieren: (1) $CHOICE_{res}$, aquellas características que se deciden positivamente o negativamente indicando que estarán presentes o no en la configuración; (2) $VARIABLE_{res}$, aquellas características que requieren asignar un valor a una variable; y (3) $CLASSIFIER_{res}$, aquellas características clonables que requieren especificar el número de instancias que se generarán. Cada selección/resolución en R resuelve exactamente una característica de V.

La formalización de un modelo de resolución R es idéntica a la formalización de V, incorporando además las siguientes definiciones:

$VSPEC_{res}$. Colección finita, de identificadores (nombres únicos) de las características (VSpecs) seleccionadas. El conjunto $VSPEC_{res}$ está particionado en $CHOICE_{res}$, $VARIABLE_{res}$, y $CLASSIFIER_{res}$. $CLASSIFIER_{res}$ incluirá todas las instancias de la característica clonable `TenantSpecific`, con un prefijo diferente para cada instancia (e.g., `MálagaTenant:TenantSpecific`). El mismo prefijo es usado para los hijos de esa instancia (e.g., `MálagaTenant:Authentication`).
$resolved : VPSEC_{res} \rightarrow VSPEC$. Función que dada una característica seleccionada en el modelo de configuración (e.g., `MálagaTenant:Authentication`), devuelve la característica original del modelo de variabilidad (e.g., `Authentication`).

$decision : CHOICE_{res} \rightarrow Boolean$. Dada una característica de tipo $CHOICE_{res}$, devuelve verdadero si la característica fue seleccionada en la configuración o falso en caso contrario.
$value : VARIABLE_{res} \rightarrow Value$. Función que dada una característica de tipo $VARIABLE_{res}$, devuelve el valor asignado a ella en la configuración.

5.3. Generación del modelo de composición

El algoritmo para generar el modelo de composición (*Create Weaving Model*) está definido en la Figura 6. El algoritmo recibe como entrada: (1) la nueva configuración evolucionada (obtenida del algoritmo *Evolve Configuration*), (2) las diferencias entre la configuración anterior y la nueva configuración evolucionada (obtenidas del algoritmo *Difference Configuration*), y (3) dos arquitecturas software; la arquitectura software de la aplicación completa ya evolucionada y la arquitectura software correspondiente a la configuración actualmente desplegada. La salida generada es un modelo en CVL con la información de los elementos arquitectónicos que tienen que ser añadidos, eliminados, y/o reconfigurados en la configuración actual de la arquitectura.

El modelo de composición es una extensión del modelo de variabilidad de CVL, que incluye la configuración a desplegar junto con los puntos de variación asociados y las transformaciones de modelo a ejecutar en cada punto de variación por el motor de CVL (`CVL execution engine`). De esta manera, primero el algoritmo extiende el modelo de configuración evolucionado para incluir las diferencias, en forma de nuevas características, en la configuración del modelo de composición (líneas 2 y 3 del algoritmo en la Figura 6). A continuación, el algoritmo genera un punto de variación por cada diferencia en la configuración (líneas 5-32) asociando el tipo apropiado de punto de variación con el tipo de cambio requerido. Por ejemplo, para sustituir un elemento (e.g., un componente) existente en la arquitectura, asociamos un punto de variación del tipo `FragmentSubstitution` (líneas 13-15). Para añadir un nuevo elemento a la arquitectura usamos un `OVP` (*Opaque Variation Point*) con una transformación de modelo encargada de componer (*weave*) el nuevo elemento (líneas 16-21), que puede ser diferente para cada característica (línea 17). Para actualizar el valor de un parámetro usamos el punto de variación `ParametricSlotAssignment` (líneas 22-23). Por último, para eliminar un elemento existente de la arquitectura usamos otro `OVP` con una transformación de modelo que realice la operación de descomposición (*unweaving*) (líneas 25-29).

Finalmente, el modelo de composición es ejecutado por CVL para generar la arquitectura evolucionada con los cambios requeridos para cada tenant.

6. Evaluación

En esta sección evaluamos la eficiencia y corrección de nuestra propuesta calculando la complejidad en tiempo de los algoritmos presentados en las Secciones 4 y 5, y modelamos los modelos de variabilidad y configuraciones como un Problema de Satisfacción de Restricciones (CSP, de *Constraint Satisfaction Problem*). Además, se proporciona una implementación Java de los algoritmos [3].

6.1. Eficiencia de los algoritmos de evolución

Para evaluar la eficiencia de los algoritmos analizaremos la complejidad según el número de operaciones básicas en función del tamaño de la entrada [3].

[3] En `http://150.214.108.91/code/cvl` y `http://150.214.108.91/code/cvltool`.

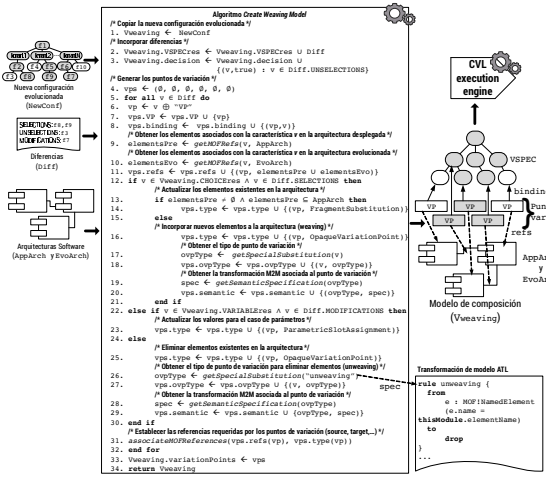


Figura 6. Algoritmo para generar el modelo de composición en CVL.

Consideramos las siguientes operaciones básicas: (i) la unión de conjuntos con un solo elemento que se corresponde con la incorporación de una característica al modelo de variabilidad o configuración; (ii) la diferencia de conjuntos con un elemento que representa la eliminación de una característica del modelo; y (iii) la comprobación de pertenencia de un elemento a un conjunto. Formalmente, sea A el conjunto de características de un modelo y x una característica dada. El tamaño de la entrada de los algoritmos de evolución es el tamaño del modelo de variabilidad (m, el número de características) y el tamaño del modelo de resolución (n, el número de características seleccionadas/resueltas). El tamaño del modelo de resolución depende del número de tenants (t) — es decir, t es el número de instancias de las características clonables en el modelo de configuración. Para simplificar, consideramos $n = m \times t$ como el peor caso, en el cual todas las posibles resoluciones para cada característica clonable han sido seleccionadas. Normalmente $n \leq m \times t$ debido a las restricciones del modelo de variabilidad.

Tabla 1. Complejidad de los algoritmos de evolución.

Algoritmo	Complejidad en tiempo
Evolve Configuration (Figura 3)	$\mathcal{O}(5n^2 + \frac{3}{t}n^2 + (9 + t + \frac{1}{t})n)$
Difference Configuration (Figura 5)	$\mathcal{O}(4n^2 + (8 + t)n)$
Create Weaving Model (Figura 6)	$\mathcal{O}(3n^3 + 18n^2 + 3n)$

Para el algoritmo *Evolve Configuration* el peor caso viene dado para las entradas con una configuración previa de tamaño $n = m \times t$ y unos nuevos requisitos de tamaño también $n = m \times t$, lo que implica cambios en todas las características. La eficiencia del algoritmo puede capturarse fácilmente siguiendo la notación $\mathcal{O}$. Por ejemplo, crear un nuevo modelo de configuración conforme

a la estructura del modelo de variabilidad evolucionado (línea 1 en la Figura 3) conlleva $\mathcal{O}(m \cdot n)$ operaciones. Copiar la configuración previa (línea 2) requiere de $\mathcal{O}((3+\frac{1}{t})n^2+t \cdot n)$ operaciones en el peor caso. Siguiendo un análisis similar para el resto del algoritmo, éste tiene una complejidad $\mathcal{O}(n^2)$. La Tabla 1 muestra la complejidad computacional para los tres algoritmos. Los dos primeros tienen una complejidad cuadrática ($\mathcal{O}(n^2)$), mientras que la complejidad del tercer algoritmo es cúbica ($\mathcal{O}(n^3)$) en términos de n.

La Figura 7 muestra los resultados de los experimentos empíricos realizados para los tres algoritmos. La eficiencia en los tres casos depende del número de tenants (t) y del número de características del modelo de variabilidad evolucionado (m). Los experimentos se llevaron a cabo en un ordenador portátil Intel Core i3 M350, 2.27GHz, 4 GB de memoria principal, y con la versión 1.7 de la máquina virtual de Java. Los resultados muestran que para evolucionar una configuración con 1000 tenants y 1000 características resueltas para cada tenant, el algoritmo *Evolve Configuration* tarda 50 segundos de media. Para calcular la diferencia entre dos configuraciones, el algoritmo tarda 45 segundos. Finalmente, crear el modelo de composición tarda alrededor de 6 minutos en el peor caso — i.e., cuando todas las características cambian, y por lo tanto, se debe definir un punto de variación para cada una de ellas. En cualquier caso, la eficiencia es aceptable para modelos de variabilidad y configuración de gran tamaño (e.g., un millón de características).

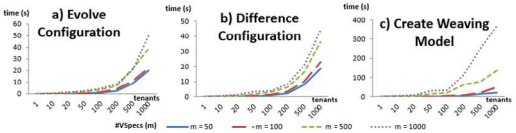


Figura 7. Eficiencia de los algoritmos de evolución.

6.2. Corrección de los algoritmos de evolución

Para demostrar la corrección de los algoritmos de evolución, hemos modelado el modelo de variabilidad y de configuración usando Choco (`http://choco-solver.org/`) una biblioteca Java para Problemas de Satisfacción de Restricciones (CSPs) [16]. Un problema CSP está definido por una tripleta (X, D, C), dónde X es el conjunto de variables, D es el dominio de las variables, y C es el conjunto de restricciones. Mapeamos el modelo de variabilidad en CVL a estos conceptos: (i) las variables son las características del modelo de variabilidad; (ii) el dominio es $\{0, 1\}$ para indicar si la característica ha sido seleccionada o no en una configuración; y (iii) las restricciones, que incluyen las relaciones padre-hijo del árbol y las restricciones OCL. Choco permite generar automáticamente un conjunto mínimo de todas las posibles configuraciones que satisfacen el conjunto inicial de restricciones. Para ello, definimos una función objetivo en CSP ($\sum_{i=1}^{n} v_i$) que minimiza el número de características seleccionadas — i.e., el número de variables con el valor 1. Se puede comprobar que la salida de los algoritmos se corresponde con una de las tuplas $v = \{v_1, \ldots, v_n\}$ generadas por Choco.

7. Trabajo relacionado

A pesar de existir multitud de trabajos centrados en modelar la variabilidad usando características clonables [7,8,17], sólo algunos de ellos tienen en cuenta el problema de la evolución de una familia de productos y los correspondientes cambios en los productos específicos [8,17]. La mayoría de los trabajos gestionan la variabilidad usando modelos de características clásicos (*feature models*) en una SPL [1,5,8], o arquitecturas de referencia [19]. Los modelos de características tienen la ventaja de ser muy conocidos y hay muchas herramientas que le dan soporte (e.g., Hydra Tool presentado en [8]). Sin embargo, el principal inconveniente de los modelos de características clásicos es que requieren un proceso adicional para establecer las relaciones entre la variabilidad especificada a nivel abstracto (e.g., en el árbol) y los puntos de variación en la arquitectura software. En [8], un lenguaje independiente para modelar la variabilidad (VML, de Variability Modeling Language) [13] es usado para establecer la correspondencia entre las características del árbol y las acciones a realizar en la arquitectura software. VML depende del lenguaje usado para modelar la arquitectura, y por lo tanto, es necesario crear manualmente un fichero VML por cada modelo de características y por cada lenguaje de modelado de arquitectura. CVL, por el contrario, facilita la propagación de los cambios a la arquitectura definiendo puntos de variación como parte del modelo de variabilidad, que además soporta cualquier arquitectura definida en lenguajes basados en MOF.

Otro punto a tener en cuenta es que la mayoría de las propuestas existentes se centran en modelar la variabilidad de propiedades no funcionales de las aplicaciones multi-tenant, como por ejemplo el precio de los servicios, su disponibilidad o satisfacción de los usuarios [5,9,14], o se centran en analizar cómo las diferentes variaciones en los servicios afectan a los atributos de calidad de la arquitectura en términos no funcionales (e.g., rendimiento, eficiencia, etc.) [19]. Aunque ambos son aspectos de gran relevancia en el desarrollo de cualquier aplicación software, ninguna de las propuestas existentes aborda el modelado de la variabilidad de los componentes funcionales de la arquitectura (ej: la variabilidad de un componente de autenticación), como proponemos en este artículo.

8. Conclusiones y trabajo futuro

En este artículo hemos presentado una propuesta que usa el lenguaje CVL y los modelos de variabilidad con cardinalidad para gestionar la variabilidad y evolución de un número elevado de tenants en el contexto de aplicaciones en la nube. Evolucionar miles de configuraciones en aplicaciones multi-tenant es una tarea intratable de realizar manualmente. Hemos definido tres algoritmos que automáticamente evolucionan una configuración previa de los tenants, calculan los cambios que se deben hacer en la arquitectura de la aplicación, y propagan dichos cambios a la arquitectura multi-tenant usando transformaciones modelo a modelo. Hemos formalizado los modelos CVL como un problema CSP para demostrar la corrección de los algoritmos y analizar la eficiencia de los algoritmos.

Nuestro trabajo futuro incluye la reconfiguración dinámica de las arquitecturas multi-tenant ejecutando los modelos CVL en tiempo de ejecución.

Agradecimientos

Trabajo financiado por los proyectos MAGIC P12-TIC1814 y HADAS TIN2015-64841-R.

Referencias

1. Abu Matar, M., Mizouni, R., Alzahmi, S.: Towards software product lines based cloud architectures. In: IEEE IC2E. pp. 117–126 (2014)
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM 53(4), 50–58 (2010), http://doi.acm.org/10.1145/1721654.1721672
3. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press (2009)
4. Bosch, J.: Design and use of software architectures: adopting and evolving a product-line approach. Pearson Education (2000)
5. Cavalcante, E., Almeida, A., Batista, T., Cacho, N., Lopes, F., Delicato, F.C., Sena, T., Pires, P.F.: Exploiting software product lines to develop cloud computing applications. In: Software Product Line Conference. pp. 179–187. SPLC (2012)
6. CVL Submission Team: Common Variability Language (CVL), OMG revised submission. http://www.omgwiki.org/variability/ (2012)
7. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specialization. SP: Improvement and Practice 10(1), 7–29 (2005)
8. Gamez, N., Fuentes, L.: Architectural evolution of famiware using cardinality-based feature models. Information and Software Technology 55(3), 563–580 (2013)
9. García-galán, J., Pasquale, L., Trinidad, P., Ruiz-Cortés, A.: User-centric adaptation analysis of multi-tenant services. ACM Trans. Auton. Adapt. Syst. 10(4), 24:1–24:26 (2016)
10. Haugen, O., Moller-Pedersen, B., Oldevik, J., Olsen, G., Svendsen, A.: Adding standardized variability to domain specific languages. In: SPLC (2008)
11. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Sci. Comput. Program. 72(1–2), 31–39 (2008)
12. Krebs, R., Momm, C., Kounev, S.: Architectural concerns in multi-tenant saas applications. CLOSER 12, 426–431 (2012)
13. Loughran, N., Sánchez, P., Garcia, A., Fuentes, L.: Language support for managing variability in architectural models. In: Software Composition (2008)
14. Mietzner, R., Metzger, A., Leymann, F., Pohl, K.: Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. In: Principles of Engineering Service Oriented Systems. pp. 18–25 (2009)
15. Pohl, K., Böckle, G., Linden, F.J.v.d.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc. (2005)
16. Tsang, E.: Foundations of constraint satisfaction, vol. 289 (1993)
17. White, J., Schmidt, D., Benavides, D., Trinidad, P., Ruiz-Cortes, A.: Automated diagnosis of product-line configuration errors in feature models. In: Software Product Line Conference. pp. 225–234 (2008)
18. Wilder, B.: Cloud Architecture Patterns: Using Microsoft Azure. O'Reilly (2012)
19. Yang, H., Zheng, S., Chu, W.C., Tsai, C.T.: Linking functions and quality attributes for software evolution. In: APSEC. pp. 250–259 (2012)

Hacia el uso de sistemas de recomendación en sistemas de alta variabilidad

Jorge L. Rodas[1], Javier Olivares[2], José A. Galindo[2], and David Benavides[3]

[1] University of Milagro, Ecuador . jrodass@unemi.edu.ec
[2] Inria - Rennes, France . {javier.olivares,jagalindo}@inria.fr
[3] University of Seville, Spain . benavides@us.es

Resumen Los sistemas de alta variabilidad son sistemas de software que describen una gran cantidad de configuraciones. Existen sistemas de alta variabilidad que representan miles de productos. Manejar la variabilidad presente en estos sistemas es costoso y en muchos casos suele ser complicado. En la actualidad hemos visto en en la industria un notable crecimiento de los sistemas de recomendación en muchos ámbitos, como el comercio electrónico, publicidad online, entre otros. Un sistema de recomendación es un agente de software que permite hacer predicciones de una serie de productos para que se adapten mejor a las necesidades o gustos de un usuario. En este artículo de prospección proponemos la fusión de estos dos campos de la ingeniería para mejorar distintas facetas dentro de la gestión de los sistemas de alta variabilidad.

Keywords: Feature models, recommender systems, variability intensive systems

1. Introducción

Los sistemas de alta variabilidad son sistemas de software cuyo comportamiento puede ser personalizado de acuerdo con las necesidades específicas de un contexto particular [1]. Un sistema de alta variabilidad puede ser representado por un modelo de características que define el número de combinaciones posibles para las configuraciones derivadas del mismo. Esta complejidad hace que el análisis y mejora de dichos sistemas de una forma manual sea una actividad costosa y propensa a errores. Para aliviar este problema se han propuesto múltiples técnicas que permiten un análisis automático de los sistemas de alta variabilidad [2]. De hecho, en la industria podemos encontrar varios ejemplos de dichos modelos que representan la variabilidad de estos sistemas en entornos reales, como el ecosistema de dispositivos móviles [6] o los sistemas de gestión de precios en la nube (cloud-price management system)[7] que describen cientos de configuraciones diferentes.

Por otra parte, en la industria encontramos los sistemas de recomendación. Un sistema de recomendación tiene como objetivo recomendar a los usuarios los productos más adecuados de acuerdo a su perfil de gustos. Normalmente son usados para sugerir los productos que mejor se adaptan a un usuario o a un tipo

de usuario, satisfaciendo sus deseos y necesidades [4] . Actualmente, estos sistemas se aplican con éxito en diferentes entornos de comercio electrónico, como por ejemplo, la recomendación de noticias, películas, música, libros, entretenimiento, entre otros.

En este artículo, vislumbramos el uso de sistemas de recomendación para las configuraciones de distintos productos así como de las características de una determinada configuración de producto, el testing de configuraciones propensas a errores y el diagnóstico personalizado. La sección 2 muestra algunos antecedentes para comprender el alcance de la propuesta. La sección 3, presenta algunas de las posibles aplicaciones de los sistemas de recomendación en el contexto de los sistemas de alta variabilidad y finalmente, en la sección 4 se concluye el trabajo.

2. Antecedentes

Un sistema de recomendación se define como un sistema que guía a los usuarios de forma personalizada en la búsqueda de productos interesantes o útiles dentro de un gran conjunto de posibilidades a elegir. En el mercado existen experiencias prácticas exitosas de la implementación de las tecnologías de recomendación en contextos de comercio electrónico, como los casos de *Amazon* [9] y *Netflix* [11], los mismos que han contribuido al desarrollo de sistemas de recomendación en nuevos campos de aplicación. En [3,4] se describen en detalle variadas aplicaciones que emplean sistemas de recomendación.

Los sistemas de recomendación se dividen en dos grupos según el método de filtrado utilizado para la generación de las recomendaciones: sistemas colaborativos y sistemas basados en contenido. Los sistemas de filtrado colaborativo [8,9] se basan en el concepto de análisis de perfiles de usuarios, en donde las recomendaciones se generan en base a los gustos de usuarios con preferencias similares. Por ejemplo, en *Movielens* (`https://movielens.org/`), un usuario que ha evaluado una serie de películas, recibirá recomendaciones de otros usuarios que hayan evaluado las mismas películas o al menos una gran parte de ellas, es decir, de aquellos usuarios con intereses similares. Los sistemas de filtrado basados en el contenido [10] son aquellos en que el usuario recibe recomendaciones basadas en sus propios gustos. En este tipo de sistemas el foco de la predicción está enmarcado en las características del perfil de usuario (para un producto se consideraría características como el precio, marca, categoría, entre otros) para hacer las recomendaciones. A modo de ejemplo, un usuario de *Amazon.com* a quién le gusta los libros de fantasía, recibirá nuevas recomendaciones de libros de ese categoría.

3. Explorando la aplicabilidad de los sistemas de recomendación

En esta sección exploramos algunas de las tareas de gestión de sistemas de alta variabilidad que podrían beneficiarse del uso de sistemas de recomendación. Para hacer más comprensible la lectura de este artículo usaremos como ejemplo un pequeño modelo de características. En los casos que se detallan a continuación, llamaremos usuario al ingeniero de pruebas y personal de operación dentro de un modelo de negocio.

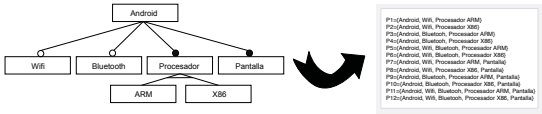


Figura 1. Modelo de características simplificado que describe la plataforma Android

Recomendación de configuraciones: Probablemente sea la aplicación más directa de los sistemas de recomendación. Consiste simplemente en recomendar productos dependiendo de configuraciones previas de otros usuarios. Por ejemplo, en la figura 1, si otros usuarios que configuraron el producto *P1:Android, Wifi, Procesador ARM*, también configuraron el producto *P2:Android, Wifi, Procesador X86*, P2 será recomendado a los que seleccionaron P1. Estas recomendaciones también se podrían realizar mediante filtros por contenido usando distintas funciones de proximidad entre características.

Recomendación de características: Este segundo caso de aplicación es similar al primero, no obstante el sistema de recomendación tomaría parte durante la configuración de un producto. Es decir, se encargaría de recomendar las características más seleccionadas por otros usuarios que ya han seleccionado la configuración parcial actual. Por ejemplo, si un usuario ha seleccionado las características *Wifi* y *bluetooth*, se le recomendaría seleccionar *Pantalla, Procesador ARM o Procesador X86*.

Recomendación de configuraciones para testing: Este último caso de aplicabilidad, representa un escenario muy interesante para los sistemas de recomendación. En este caso lo que se persigue es la recomendación de los productos que sean más susceptibles de tener errores. Para esto, partimos de la hipótesis de que los productos con menor valoración por parte de los usuarios finales, son aquellos con tendencia a contener más errores. Entonces, a partir de este conjunto de valoraciones sobre configuraciones en un modelo de características y el uso de algoritmos de filtrado colaborativo, podríamos recomendar al ingeniero de pruebas, una lista con los productos más óptimos para el testing.

Recomendación basado en el diagnóstico personalizado: Una de las posibles aplicaciones en el contexto de los sistemas de alta variabilidad es en la actividad de explicación de errores. Actualmente, la mayoría de las propuestas existentes para explicar errores no permiten optimizar el resultado de los mismos. Éstos tienden a mostrar la explicación mínima (aquella que requiere del menor número de cambios para solucionar el error). No obstante, esta explicación mínima no siempre es la más interesante [5]. Entendemos que mediante el uso de sistemas de recomendación se podrían obtener aquellas explicaciones que ayuden de una manera óptima al modelador del sistema de variabilidad.

4. Conclusiones

En este artículo de prospección proponemos el uso de sistemas de recomendación para la mejora de sistemas de alta variabilidad en casos prácticos concretos de aplicación. Para el proceso de pruebas se emplearán reportes de usuario tales como los reportes de errores (ingenieros de pruebas) o valoraciones (usuarios

finales) extraídas de bug-trackers, tiendas de apps u otros. Esta información permitirá construir un ambiente de pruebas y ejecutar experimentos que validen la propuesta. Para evaluar la propuesta se tiene contemplado tomar un conjunto de datos y contrastar los resultados obtenidos por el sistema de recomendación versus los obtenidos aplicando un mecanismo manual.

Teniendo en cuenta la importancia de los sistemas de recomendación en la actualidad, y las múltiples líneas de investigación generadas en el área, nuestros trabajos futuros pretenden abordar los siguientes temas:

- Estudiar otros métodos de ponderación de características para los algoritmos de recomendación que permitan descubrir interacciones entre varias características de acuerdo a la valoración que otorga el usuario a los productos.
- Combinar los métodos empleados en los casos presentados en esta propuesta con otros métodos de recomendación con el objetivo de mejorar las recomendaciones que se otorga al usuario (ingeniero de pruebas o usuario final).

Referencias

1. Muhammad Ali Babar, Lianping Chen, and Forrest Shull. Managing variability in software product lines. *Software, IEEE*, 27(3):89–91, 2010.
2. David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 2010.
3. Jesus Bobadilla, Antonio Hernando, Fernando Ortega, and Jesus Bernal. A framework for collaborative filtering recommender systems. *Expert Systems with Applications*, 38(12):14609 – 14623, 2011.
4. Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.
5. Alexander Felfernig, Gerhard Friedrich, Monika Schubert, Monika Mandl, Markus Mairitsch, and Erich Teppan. Plausible repairs for inconsistent requirements. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 791–796, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.
6. José A Galindo, Hamilton Turner, David Benavides, and Jules White. Testing variability-intensive systems using automated analysis: an application to android. *Software Quality Journal*, pages 1–41, 2014.
7. Jesús García-Galán, Omer F. Rana, Pablo Trinidad, and Antonio Ruiz-Cortés. Migrating to the cloud: a software product line based analysis. In *CLOSER*, 2013.
8. Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.
9. Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.
10. Michael Pazzani and Daniel Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine learning*, 27(3):313–331, 1997.
11. Alex Tuzhilin, Yehuda Koren, Jim Bennett, Charles Elkan, and Daniel Lemire. Large-scale recommender systems and the netflix prize competition. In *KDD Proceedings*, 2008.

Measuring the quality of transformation alternatives in software architectures evolution

Javier Criado[1], Silverio Martínez-Fernández[2],
David Ameller[2] and Luis Iribarne[1]

[1] Applied Computing Group, University of Almería, Spain
{javi.criado,luis.iribarne}@ual.es
[2] GESSI Research Group, Universitat Politècnica de Catalunya, Barcelona, Spain
{smartinez,dameller}@essi.upc.edu

Abstract. Many today's software systems need to be self-adapted at run-time. Model transformation is a good approach to adapt the component-based architecture of such software systems. However, existing model transformation processes focus on the functionalities of systems, giving less importance to the quality attributes. The goal of this study is to improve model transformation processes by also considering quality attributes in the generation and adaptation of component-based architectures (*i.e.*, driving the selection among many alternative model transformations by software architecture metrics). Such metrics evaluate the qualities of an architecture, such as flexibility and modifiability. This paper provides some measures of quality for different transformation alternatives and an example in the ENIA software.

Keywords: component-based software, architecture configuration, architecture metrics, quality-driven transformation, model transformation.

1 Introduction

Nowadays, component-based software systems need to be self-adapted at run-time. Previous studies have shown that model transformation is a good approach to adapt the component-based architecture of software systems [4]. However, existing model transformation processes focus on the functionalities of systems, giving less importance to the Quality Attributes (QA), also known as non-functional requirements or *-ilities*. An exception are the quality-driven model transformations described in [7], in which quality is introduced on the design of the transformation process, or the proposed mechanisms related to the Dynamic Software Product Lines (DSPLs) [8] considering some quality information.

This paper presents an approach to adapt and evolve component-based software systems by measuring the quality of different transformation alternatives. The goal of this study is to improve model transformation processes by also considering QAs in the generation and adaptation of component-based architectures. As an example, this paper will focus on *flexibility* metrics to show the suitability of our QA-based transformation approach. We provide an exemplar

scenario for the ENIA (ENviromental Information Agent) software. ENIA is a geographic information system whose User Interfaces (UI) are based on coarse-grained components and need to be adapted at run-time depending on user interactions, system requirements, or other evolution needs [1].

2 QA-based Transformation Approach

Depending on a system's targeted QAs and goals (*e.g.*, improve its flexibility, maximize the modifiability, minimize the cost, or optimize the execution performance), architectural design decisions can be oriented in different ways. In this sense, decisions about the construction of software architectures, such as component selection, may differ from each other. For this reason, it is important to include QAs as part of the rationale to make such architectural decisions. The use of specific QA metrics allows us to analyze different alternatives of component configurations, and how each one of these alternative architectures is related to the QA considered.

In order to demonstrate the feasibility of this approach, we focus on some metrics related to the scenario of ENIA UIs. Nevertheless, these measurement actions can be extended and applied to other UI applications offering its functionality as a mashup or a dashboard, *e.g.*, Geckoboard, Cyfe, and Netvibes.

In the ENIA scenario, we discussed the relevant QAs: flexibility, modifiability, analyzability, performance, testability and consumed resources. Due to the limitation of space, this paper focuses on the highest priority of ENIA: the *flexibility* attribute [5] of the *quality in use model* of the ISO/IEC 25000 standard [6]. ENIA UIs (intended for managing geographic data, social network information, third-party widgets, etc.) try to provide a friendly interaction by accomplishing the following objectives: (a) user interfaces must be elastic and flexible with the aim of allowing the modification of their structure; (b) users can reconfigure and customize their interfaces, *e.g.*, resizing the component displayed in the interface, changing its position, adding new components, and removing existing ones; and (c) user interfaces with a greater number of simple components are preferred over interfaces with a lower number of complex components gathering a large amount of functionality (*i.e.*, the more pieces has an interface, the more reconfiguration operations can be performed on it).

The metrics addressed to measure the flexibility of ENIA UIs are calculated from the component and architecture specifications (*i.e.*, information provided by developers, architects, and experts) and used at run-time. This paper proposes an initial stage of the work based on two simple derived metrics. Simple and realistic metrics allow easier adoption in industry. The first metric is related to the number of components. The second metric is related to the *isResizable* property of the specification, which indicates whether a component can be resized or not. Table 1 shows the information about both metrics (rc and rr).

The values obtained from these metrics are used for choosing the best alternative of architecture that can be constructed from a previous one. Both values (rc and rr) must be maximized and, therefore, a trade-off must be performed.

This information is intended to be used not only for guiding the design and development of UIs, but also to be applied for supporting the adaptation of this type of architectures at run-time [4].

Figure 1 shows an example of the measurement that can be performed in the context of the ENIA system. Consider the initial UI provided by ENIA prototype implementation [1]: a software architecture representing an UI made up from two components, a map (M) for displaying geographical information layers, and a component showing the messages of a social network account (T). From this UI, if we want to incorporate a legend (L) and a list of layers (LL) related to the map, it is possible to choose among three alternatives. In this example, the number in labels represent different alternatives of the same component type (*e.g.*, M1, M2, and so on). M3 gathers the functionalities of M, L and LL (which is represented by its three provided interfaces). C1 acts as a container delegating the interfaces of M2, L2 and LL2. Moreover, M1, M3, T1, L1, and C1 are resizable whereas M2, L2 and LL2 are not. The values obtained for rc and rr shows that A12 has the best value of rc, but A13 has the best value of rr. Nevertheless, A11 has the best value for the average of rc and rr, 0.775 instead of the 0.7 of A12 and A13.

The transformations to get these alternatives are different. The transformation for adapting A0 to A11 (T1) implies the addition of L1 and LL2 and their connection to M1 whereas A13 is obtained from the replacement of M1 by M3 (*i.e.*, performing T3 transformation). Therefore, if we intend to maximize the

Metric	Description	Derived	Exp.
c	Number of components	n	—
r	Number of resizable components	n	—
m	The highest c from all architectural alternatives	y	$max(c_1, ..., c_n)$
rc	Ratio of components according to m	y	$rc = c/m$
rr	Ratio of resizable components	y	$rr = r/c$

Table 1. Example metrics for maximizing the flexibility in ENIA user interfaces

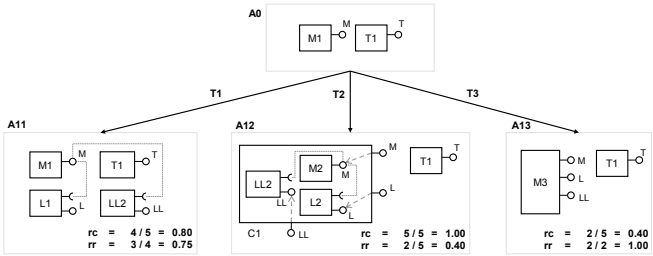


Fig. 1. Example of transformation alternatives

flexibility, T1 can be chosen as the best transformation operation for adapting the architecture of this example scenario.

3 Further Considerations

It is well accepted in the software architecture community that QAs are the most important drivers of architecture design [2]. Therefore, QAs should guide the selection of the best transformation alternative, considering the synergies and conflicts among them [3].

This research in progress aims to analyze whether considering QAs can improve model transformation processes. To that end, we have proposed metrics to measure the *flexibility*, and used them for the model transformation process. It is important to note that the proposed metrics are just an indicator of a QA, and their improvement must not be seen as a complete satisfaction of any QA.

Future work spreads in several directions: (a) proposing more metrics for more QAs considering the complexity of direct and derived metrics: modifiability (*e.g.*, modular designs, dependability of components), analyzability (*e.g.*, data about the errors of the components), performance (*e.g.*, response time), testability (*e.g.*, testing components before applying the model transformation), and consumed resources (*e.g.*, memory); (b) considering constraints in the alternative solutions (*e.g.*, regarding components' technology, location, and provider); (c) assisting software architects to reason about trade-offs on different quality attributes, so that they can prioritize; and, (d) deriving and combining metrics.

Acknowledgments. This work was funded by the Spanish MINECO and the Andalusian Government under TIN2013-41576-R and P10-TIC-6114 projects.

References

1. ACG. ENIA Poject – Development of an intelligent web agent of environmental information, 2011. `http://acg.ual.es/projects/enia/`.
2. Ameller, D., Ayala, C., Cabot, J., Franch X.: Non-functional requirements in architectural decision making. IEEE Software. 30(2), 61–67 (2013)
3. Boehm, B.: Architecture-based quality attribute synergies and conflicts. In: SAM'2015, pp. 29–34. IEEE Press (2015)
4. Criado, J., Rodríguez-Gracia, D., Iribarne, L., Padilla, N.: Toward the adaptation of component-based architectures by model transformation: behind smart user interfaces. Software: Practice and Experience 45(12), 1677–1718 (2015)
5. Herrera, M., Moraga, M.Á., Caballero, I., Calero, C.: Quality in Use Model for Web Portals (QiUWeP). In: ICWE 2010, LNCS 6385, pp, 91–101. Springer (2010)
6. ISO/IEC. ISO/IEC 25000. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE (2014)
7. Loniewsli, G., Borde, E., Blouin, D., Insfran, E.: An Automated Approach for Architectural Model Transformations. In: Information System Development: Improving Enterprise Comm., pp. 295–306. Springer International Publishing (2014)
8. Morin, B., Barais, O., Jézéquel, J.M., Fleurey, F., Solberg, A.: Models@Run.time to Support Dynamic Adaptation. Computer 42(10), 44–51 (2009)

El uso de modelos de características con atributos para pruebas en sistemas de alta variabilidad: primeros pasos

Mariuxi Vinueza[1], Jorge L. Rodas[1], José A. Galindo[2], and David Benavides[3]

[1] University of Milagro, Ecuador . {mvinuezam,jrodass}@unemi.edu.ec
[2] INRIA - Rennes, France . jagalindo@inria.fr
[3] University of Seville, Spain . benavides@us.es

Resumen Los modelos de características con atributos representan todos los productos de una línea de productos junto con información adicional. En la literatura encontramos modelos representando miles de productos distintos. La selección de estos productos para hacer pruebas es un reto que se está estudiando en la literatura, en algunas de estas propuestas utilizan modelos de características con atributos para seleccionar este subconjunto de productos. Sin embargo no existe una guía de como utilizar los atributos para selección de casos de pruebas en distintos escenarios, con el objetivo de alimentar esa guía, nos proponemos buscar en la literatura la manera de caracterizar los modelos usados por otros investigadores con el objetivo de ayudar a modelar atributos en modelos de características para realizar las pruebas.

Keywords: Modelo de características, atributos, pruebas

1. Introducción

Una línea de productos permite desarrollar un conjunto de productos distintos que comparten parte de la funcionalidad. Los *modelos de características* son usados para representar el conjunto de productos de software en términos de características y relaciones [6].

El análisis automático de modelos de características trata de extraer información de los modelos de características usando mecanismos automatizados [2]. Por ejemplo, para validar la corrección del modelo o identificar el conjunto de característcas que lo conforman. FAMA [3] o FaMiLiar[1], entre otras, son herramientas que implementan e integran diferentes soluciones para análisis automático de modelos de características.

Existen propuestas para representar la información de calidad dentro de los modelos de características. Actualmente, a este tipo de modelos con información adicional se les denomina como *extendidos, avanzados, o modelos de características con atributos* [2,4,5]. Un modelo de características extendido contiene información extra sobre las características, el propósito de esta extensión es añadir información medible (cuantitativa) sobre las características mediante la introducción de atributos, así como la inclusión de restricciones más complejas entre las características y sus atributos.

1.1. Hacia un catálogo de Modelos de Características Extendido

Las pruebas en líneas de producto software son muy complejas porque requieren la prueba de un conjunto de productos en lugar de un solo producto [4]. Esto hace que se incremente costo y tiempo cuando se realizan las pruebas. Uno de los objetivos de las pruebas de línea de producto software es seleccionar la menor cantidad de productos para encontrar la mayor cantidad de errores mientras se reduce el esfuerzo de la prueba.

Con el fin de optimizar el número de pruebas para ejecutar en una línea de productos, se necesitan mecanismos asistidos por ordenador para describir los productos válidos que pueden ser utilizados en las pruebas. Con el fin de describir los productos válidos y no válidos en una línea de productos, los investigadores utilizan una variedad de métodos de modelado de variabilidad.

Un enfoque en las pruebas en los modelos de características esta basado en atributos de características, no obstante la escazes de los modelos extendidos hace que los investigadores no cuenten con información suficiente o una guía para seleccionar los atributos correctamente.

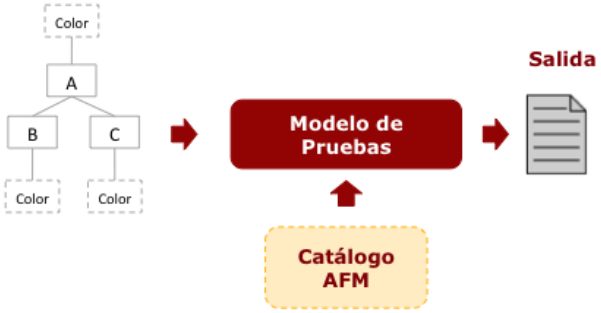


Figura 1. Modelo de pruebas.

Como se observa en la figura 1 en la creación de un modelo de pruebas se verifican los productos y atributos resultantes del proceso para maximizar la probabilidad de detectar errores mientras se reduce el esfuerzo de las pruebas. Como soporte al modelo de pruebas, el catálogo de modelos de características extendido servirá como una guía para seleccionar correctamente los atributos en diferentes casos de pruebas y luego aplicar cualquier función y poder obtener el subconjunto de productos a ensayar.

En este artículo queremos presentar los primeros avances hacia una revisión sistemática de la literatura para encontrar información sobre modelos de

características extendidos con la finalidad de crear un catálogo que facilite la prueba de las herramientas de análisis automático y además una vía para poder caracterizar los modelos existentes en la literatura.

2. Método de revisión y primeros resultados

Para la revisión de la literatura nos hemos basado en un método sistemático y estructurado [7] con el fin de obtener estudios que proporcionen información de los modelos de características extendidos.

Con la base del protocolo de revisión [7] se definieron los siguientes parámetros para la selección de los artículos:

2.1. Preguntas de investigación

Para cumplir con lo antes mencionado uno de nuestros objetivos será responder las siguientes preguntas de investigación:

- RQ1:¿Qué modelos de características con atributos se han propuesto en la literatura?
- RQ2:¿Cómo caracterizar los modelos de características con atributos?
- RQ3:¿En qué foros de investigación se publican?

2.2. Proceso de búsqueda

Para el proceso de búsqueda de documentación se definieron los siguientes datos:

- Base de datos: *SCOPUS*
- Cadena de búsqueda: *('feature model' OR 'feature diagram' OR 'feature models' OR 'feature diagrams') AND (attributes OR attributed OR quality OR extended)*. Filtros: *Ingeniería, Ciencias de la Computación y Matermática*
- Fecha de la búsqueda: 15-marzo-2016
- Años objeto de la búsqueda: a partir de 1990

2.3. Criterios de inclusión y exclusión

Artículos que cumplan los siguientes criterios, en la primer revisión se descartaron los libros y conferencias, además publicaciones que de acuerdo a nuestra experiencia el título y el resumen no estaban relacionados con modelos de características. Para una segunda revisión se seleccionarán específicamente los artículos que contengan un modelo de características extendido que puede estar representado en forma gráfica o en lista de configuraciones.

2.4. Síntesis de los datos extraídos

La búsqueda ha dado como resultado inicial 427 artículos. A este resultado se aplicó criterios de inclusión y exclusión expuestos en la sección anterior, de ese proceso resultaron descartados 303 artículos, hasta el momento disponemos de 125 artículos. Nuestra intención es poder obtener información de la mayor cantidad de modelos de características extendidos que serán la base para la creación de un repositorio.

3. Trabajo futuro y perspectivas

En esta sección nosotros presentamos nuestro trabajo futuro relacionado con los modelos de características extendidos y su aplicación.

1. Formalmente definir el porcentaje de atributos reales que hay en los modelos de características extendidos y el tipo de dominio que contemplan.
2. Analizar otros estudios similares sobre modelos que no sean modelos de características, como modelos UML.
3. Hasta que punto los modelos de características son usados realmente en la práctica y no solo en la literatura.
4. Desarrollar una herramienta para proporcionar modelos de características con atributos que puedan ser utilizados por otros investigadores.
5. Fortalecer el proceso de búsqueda de información utilizando otras bases de datos como ACM o IEEE, además incluyendo en la cadena otras variables a considerar como por ejemplo "variability".

Referencias

1. Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B France. Familiar: A domain-specific language for large scale management of feature models. *Science of Computer Programming*, 78(6):657–681, 2013.
2. David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 2010.
3. David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz Cortés. Fama: Tooling a framework for the automated analysis of feature models. *VaMoS*, 2007:01, 2007.
4. José A Galindo, Hamilton Turner, David Benavides, and Jules White. Testing variability-intensive systems using automated analysis: an application to android. *Software Quality Journal*, pages 1–41, 2014.
5. Jesús García-Galán, Omer F. Rana, Pablo Trinidad, and Antonio Ruiz-Cortés. Migrating to the cloud: a software product line based analysis. In *CLOSER*, 2013.
6. Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.
7. Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004.

Adaptando Github para el desarrollo de LPS: modelo de branching y operaciones de repositorio para los ingenieros del producto

Leticia Montalvillo y Oscar Díaz

ONEKIN Research Group.
Universidad del País Vasco (UPV/EHU),
Facultad de Informática de San Sebastián (España)
{leticia.montalvillo,oscar.diaz}@ehu.eus

Abstract. El desarrollo de las LPS se divide en dos procesos: la ingeniería del domino (ID) y la ingeniería del producto (IP). Aunque los dos procesos tienen ciclos de vida independientes, deben sincronizar sus desarrollos regularmente. Para ello, existen dos *vías*: la *actualización* (propagar artefactos de la ID a la IP) y la *retroalimentación* (propagar artefactos de la IP a la ID). Este trabajo estudia cómo soportar las *vías de sincronización* en los Sistemas de Control de Versiones (SCVs) valiéndonos de las operaciones tradicionales de los SCV (i.e., *merge*, *branch*, *fork* y *pull*). De este modo, las divergencias que surgen en la sincronización se pueden resolver *à la VCS*, es decir, destacando las diferencias entre las distintas versiones de un mismo artefacto. Sin embargo, esto se traduce en una brecha conceptual entre cómo se conciben las propagaciones (i.e., *actualización*, y *retroalimentación*) y cómo se realizan las propagaciones (i.e., *merge*, *branch*, etc.). Para reducir esta brecha, proponemos adaptar los SCV existentes para que las *vías de sincronización* de las LPS sean operaciones de primera clase. Como prueba de concepto, utilizamos técnicas de Aumentación Web para añadir esta funcionalidad a las páginas web de Github. A través de un solo clic, los ingenieros del producto pueden ahora (1) generar repositorios de producto, (2) actualizar el producto con nuevas versiones de los artefactos reutilizables, o (3) retroalimentar la LPS promocionando artefactos específicos de producto a artefactos reutilizables.

Publicado en

ACM Proceedings of the International Conference on Software Product Line (SPLC), pp. 111-120, 2015
http://dl.acm.org/citation.cfm?doid=2791060.2791083

FLAME: a formal framework for the automated analysis of software product lines validated by automated specification testing

Amador Durán, David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés

Deparment of Computer Languages and Systems - ISA gruop - University of Seville
{amador,benavides,sergiosegura,ptrinidad,aruiz}@us.es

Resumen En una revisión de la literatura de los últimos 20 años sobre análisis automático de modelos de características, la formalización de las operaciones de análisis fue identificado como uno de los retos principales en el área. Esta formalización podría ofrecer resultados interesantes para los desarrolladores de herramientas como una definición precisa de las operaciones de análisis y, aún más importante, una implementación de referencia que aun no siendo necesariamente eficiente en términos computacionales pueda servir para comparar la salida de distintas herramientas. En el artículo se presenta el framework FLAME como resultado de afrontar este desafío. FLAME es un framework formal que puede ser usado para especificar, no solo la semántica de los modelos de características sino también de otros modelos de variabilidad como OVM. Par ello se ha diseñado una arquitectura en dos niveles. El nivel abstracto que es el nivel en el que se especifica el conjunto de operaciones de análisis independientemente de cualquier lenguaje de variabilidad concreto. Sobre este nivel, se pueden anclar distintos niveles para cada lenguaje específico de variabilidad redefiniendo algunos tipos y relaciones abstractas definidas en el nivel inferior. La verificación y validación de FLAME ha seguido un proceso en el que se han realizado verificaciones formales de manera tradicional por medio de pruebas manuales de teoremas mientras que la validación ha sido hecha integrando nuestra experiencia en pruebas metamórficas sobre herramientas de análisis de modelos de variabilidad. Para seguir esta validación guiada por casos de prueba, la especificación de FLAME, escrita en Z, fue traducida a Prolog y 20,000 casos de prueba aleatorios fueron generados y ejecutados automáticamente. Los resultados de la ejecución de los casos de prueba permitió descubrir algunas inconsistencias no solo en la especificación formal sino también en las definiciones informales previas de las operaciones de análisis.

Keywords: especificaciones formales, pruebas de especificación, líneas de producto software, modelos de carateríticas

Probando sistemas altamente configurables mediante análisis automático de modelos de características: El caso de Android

José A. Galindo[1], Hamilton Turner[2], David Benavides[1], and Jules White[3]

[1] University of Seville, Spain . {jagalindo,benavides}@us.es
[2] Virginia Tech, USA. hamiltont@vt.edu
[3] Vanderbilt University, USA. jules@dre.vanderbilt.edu

Resumen Los modelos de características surgieron para representar las partes comunes y variables de los sistemas de alta variabilidad. Asimismo, el elevado número de configuraciones existentes en un sistema de alta variabilidad imposibilitan el análisis manual de los mismos. Para atacar el problema existen propuestas que se centran en reducir el espacio de pruebas a considerar. No obstante, es importante modelar el coste y valor de las pruebas para poder optimizar las ganancias obtenidas durante el proceso de prueba. En este artículo presentamos TESting vAriAbiLity Intensive Systems (TESALIA), una propuesta que usa el análisis automático de modelos de características para la optimización de la generación de pruebas. Concretamente, se propone representar el valor y el coste de las pruebas usando atributos de calidad y modelarlos como un problema de satisfacción de restricciones. Con esto se consigue reducir, priorizar y empaquetar distintos casos de prueba mientras que se respetan ciertas restricciones de negocio. Asimismo, se ofrece un prototipo con la implementación de TESALIA y se valida la propuesta en el contexto de pruebas para aplicaciones móviles Android, donde mostramos los beneficios de maximizar la cuota de mercado cubierta por nuestras pruebas a la vez que respetamos un gasto máximo.

Keywords: Modelo de características, atributos, pruebas

Validación de un Método Dirigido por Modelos para la Evaluación y Mejora de Arquitecturas Software: Una Familia de Experimentos

Javier González-Huerta[1], Emilio Insfrán[1], Silvia Abrahão[1], Giuseppe Scanniello[2]

[3]Departmento de Sistemas Informáticos y Computación
Universitat Politècnica de València, España
{jagonzalez, einsfran, sabrahao}@dsic.upv.es
[2]Dipartimento di Matematica, Informatica e Economia, Universidad de Basilicata, Italia
giuseppe.scanniello@unibas.it

Resumen. *Contexto:* Las arquitecturas software deben ser evaluadas durante las primeras fases del desarrollo de software con el fin de verificar si los requisitos no funcionales (RNFs) del producto se pueden cumplir. Esta actividad es aún más crucial en el desarrollo de Líneas de Producto Software (LPS), ya que también es necesario identificar si los RNFs de un producto en particular se puede lograr mediante la utilización de los mecanismos de variación proporcionados por la arquitectura de la línea de productos o si se requieren transformaciones arquitectónicas adicionales. Estas cuestiones nos han motivado a proponer QuaDAI, un método para la derivación, evaluación y mejora de arquitecturas software para el desarrollo dirigido por modelos de líneas de producto.

Objectivo: Este artículo presenta los resultados de una familia de cuatro experimentos llevados a cabo para validar empíricamente la estrategia de evaluación y mejora de QuaDAI.

Método: La familia de experimentos se llevó a cabo por 92 evaluadores noveles: alumnos de grado y máster en Ingeniería del Software de la Universidad Politècnica de València y la Universidad de Basilicata en Italia. El objetivo fue comparar la efectividad, la eficiencia, la facilidad de uso percibida, la utilidad percibida y la intención de uso de los participantes utilizando QuaDAI en comparación con el método ATAM (*Architecture Tradeoff Analysis Method*).

Resultados: Los resultados mostraron que los participantes han realizado la evaluación de las arquitecturas más rápido y han obtenido arquitecturas software de mayor calidad con el método QuaDAI. Los participantes también han encontrado este método más fácil de usar, más útil y además han expresado una mayor intención de uso que el método ATAM. Los resultados del meta-análisis llevado a cabo a partir de la agregación de los resultados obtenidos en los experimentos individuales también confirman estos resultados.

Conclusiones: Los resultados confirman la hipótesis de que QuaDAI lograría mejores resultados que ATAM con evaluadores noveles ya que minimiza la subjetividad de este método y que QuaDAI puede ser considerado un enfoque prometedor para apoyar evaluaciones arquitectónicas que se producen tras la derivación de la arquitectura del producto en procesos de desarrollo de líneas de producto dirigido por modelos.

Calidad y Pruebas

Calidad y Pruebas

Coordinadores: Carme Quer y María José Suárez-Cabal

Lorena Gutiérrez-Madroñal, Inmaculada Medina-Bulo y Juan José Domínguez Jiménez. *Generación automática de eventos de prueba para sistemas de IoT*. (Completo)

María José Escalona, Guillermo Lopez, Sira Vegas, Laura García-Borgoñón, Julián Alberto García García y Natalia Jurista. *A Software Engineering Experiments to value MDE in testing. Learning Lessons*. (Completo)

Raquel Blanco y Javier Tuya. *Pruebas sobre aplicaciones de bases de datos orientadas a grafos: un enfoque basado en modelos*. (Completo)

Jesús Morán, Claudio De La Riva y Javier Tuya. *Generación y Ejecución de Escenarios de Prueba para Aplicaciones MapReduce*. (Completo)

Khalid Alkharabsheh, Yania Crespo, M. Esperanza Manso y José Ángel Taboada. *Sobre el grado de acuerdo entre evaluadores en la detección de Design Smells*. (Completo)

Khalid Alkharabsheh, Yania Crespo, M. Esperanza Manso y José Ángel Taboada. *Comparación de herramientas de Detección de Design Smells*. (Completo)

Manuel I. Capel, Anna Grimán Padua y Eladio Garví García. *Calidad Ágil: Patrones de Diseño en un contexto de Desarrollo Dirigido por Pruebas*. (Completo)

Abel Lorente Ramírez, Ignacio García-Rodríguez de Guzmán y Mario Piattini Velthuis. *Hacia un entorno extensible basado en ADM para la refactorización de sistemas heredados*. (Corto)

Antonio Estero-Botaro, Francisco Palomo-Lozano, Inmaculada Medina-Bulo, Juan José Domínguez-Jiménez y Antonio García-Domínguez: *Quality metrics for mutation testing with applications to WS-BPEL compositions*. Software Testing, Verification and Reliability, Volume 25, Issue 5-7, 536–571, 2015 (DOI: 10.1002/stvr.1528). (Relevante)

Rubén Casado, Javier Tuya y Muhammad Younas: *Evaluando la efectividad de un modelo abstracto de transacciones para la prueba de transacciones en servicios web*. Concurrency and Computation: Practice and Experience, 27(4), 765–781, 2015. (DOI: 10.1002/cpe.2851). (Relevante)

Generación automática de eventos de prueba para sistemas de IoT*

Lorena Gutiérrez Madroñal, Inmaculada Medina Bulo, and Juan José Domínguez Jiménez

Grupo de Investigación UCASE de Ingeniería del Software.
Universidad de Cádiz
{lorena.gutierrez,inmaculada.medina,juanjosedominguez}@uca.es

Resumen La aplicación en diversas áreas de Internet de las Cosas (IoT) ha ido en aumento en los últimos años. Uno de los principales inconvenientes que tienen los sistemas IoT es la cantidad de información que tienen que manejar. Esta información llega en forma de eventos, cuyo receptor ha de tomar las decisiones correctas, en tiempo real, según los datos recibidos. Viendo la relevancia que tiene el procesado de esta información, resulta fundamental analizar y probar los sistemas IoT que van a trabajar con ella. Para probar las distintas funcionalidades de los sistemas IoT, se necesita una gran cantidad de eventos con estructuras y valores específicos. Conseguir estos eventos de forma manual puede ser una tarea muy costosa y propensa a errores. En este trabajo se presenta un método para la generación automática de eventos de prueba para sistemas de IoT. Los resultados obtenidos en los casos de prueba utilizados muestran su viabilidad.

1. Introducción

Hoy en día la información se ha vuelto vital para tomar decisiones. La información es especialmente crucial en sectores comerciales, para detectar fraudes, para manejar datos de mercado, las redes de sensores... En el 2008 Haller et al. [1] definieron IoT como "Un mundo donde los objetos físicos que se integran en la red se convierten en participantes activos en los procesos de negocios". En este mundo se maneja, en tiempo real, un gran volumen de información que llega de diversas fuentes en forma de mensajes o eventos. Como consecuencia se han desarrollado diversas herramientas, dispositivos y mecanismos para obtener, procesar y transmitir esta información. Uno de los mayores problemas de estos sistemas es monitorizar la información y responder con la menor latencia. Para intentar solventarlo David Luckham propone el *Procesado de Eventos Complejos* (CEP) [2].

A raíz de trabajar con CEP, Luckham plantea el uso de lenguajes de alto nivel para detectar *patrones de eventos* y *reglas*. Donde los *patrones de eventos*

* Este trabajo ha sido financiado por el proyecto DArDOS TIN2015-65845-C3-3-R del Programa Nacional de Investigación, Desarrollo e Innovación del Ministerio de Economía y Competitividad.

son plantillas en las que se reemplazan sus variables por los valores de los eventos y las *reglas* son, en su conjunto, un método para procesar eventos. Años más tarde Schiefer et al. [3] presentan *Event Processing Language* (EPL), un lenguaje estandarizado para definir patrones de eventos y reglas, el cual se ha ido especializando dando lugar a varios EPLs con diferentes propósitos y características, pero siempre con una cosa en común, todos trabajan con eventos.

Profundizando en los conceptos del mundo IoT, hay que rescatar dos conceptos que han de tenerse en cuenta y que fueron definidos por Dunkel et al. [4]:

- **Instancias de eventos**: Toda situación que requiera una reacción por parte del sistema.
- **Tipos de eventos:** Clasificación de las instancias de evento según sus características, es decir, cada instancia de evento pertenece a un *tipo de evento*.

El concepto de *tipos de eventos* fue expresado con mayor detalle por Luckham en [5], donde exponía que: "Si la actividad que queremos procesar puede representarse por más de un evento, de cada uno se recogerán diferentes atributos. Si el **atributo de un evento** es un componente de la estructura de un evento, un **tipo de evento** se crea dependiendo de los atributos que hemos guardado, por lo tanto, la colección de los atributos de un evento determinan su estructura".

A medida que hemos ahondado en el mundo IoT comprobamos la importancia de los eventos, sus atributos y, como consecuencia, los tipos de eventos. Es por esto por lo que nos resulta esencial analizar el comportamiento de los sistemas IoT que van a trabajar con ellos. Por este motivo, para poder realizar diferentes tipos de pruebas a estos sistemas: testear partes específicas de la consulta, provocar situaciones extremas o errores, necesitamos trabajar con los valores de los eventos. El artículo se organiza de la siguiente forma: en la Sección 2 se recogen trabajos relacionados, en la Sección 3 se presenta el método con el que se obtienen eventos para pruebas, método que se prueba en diferentes situaciones presentándose los resultados obtenidos en la Sección 4 con diferentes casos de prueba y finalmente en la Sección 5 las conclusiones y el trabajo futuro.

2. Trabajos relacionados

Comenzamos esta sección resaltando aquellos trabajos que están enfocados en hacer pruebas en aplicaciones CEP, cada uno se centra en un tipo diferente de problema o situación de estos sistemas.

Habibi et al. [6] presentan un procedimiento de seis etapas para testear aplicaciones web que manejan eventos. Las pruebas en este tipo de sistemas, cuyo comportamiento se determina por los eventos, necesitan una gran cantidad de eventos que cubran las diferentes etapas. Su procedimiento tenía tanto ventajas como desventajas, estas últimas causadas por el uso de la prueba de mutaciones, ya que esta incrementaba el tiempo de las pruebas.

El objetivo del sistema de código abierto QCEP-TS y su lenguaje EPTDL [7], es hacer pruebas de las funcionalidades CEP, entre las que se incluyen pruebas de activación por tiempo en un entorno simulado. Ellos consideran que nombrar

de forma adecuada los eventos y sus atributos, ayudan a que sus definiciones sean más comprensibles.

FINCoS [8,9] es una herramienta que intenta solventar algunos problemas de los sistemas CEP: la falta de estándares, sus múltiples dominios de aplicación y las métricas. Sus autores piensan que FINCoS puede ayudar a mejorar los motores CEP, y a desarrollar nuevos benchmarks.

Otros estudios se centran en el benchmarking de sistemas CEP [10,11], donde presentan un benchmark CEPBen, que evalúa los comportamientos funcionales de CEP. CEPBen está implementando sobre la plataforma Esper [12] y ha sido usado para explorar los factores de rendimiento y nuevas métricas en la gestión del rendimiento de los sistemas CEP. Los resultados obtenidos de las pruebas de rendimiento demuestran la influencia de los comportamientos funcionales en el rendimiento del sistema CEP.

Finalmente, un repaso sobre los generadores de eventos existentes nos revela que algunos son de propósito comercial [13,14], otros solo pueden ser aplicados en aplicaciones específicas ofertadas por los autores [15], y otros se centran en la generación de eventos de áreas muy específicas [16,17,18], como condiciones ambientales. Dada la necesidad de información por el "usuario de a pie", los generadores de eventos dieron paso a las plataformas IoT que son más accesibles y fáciles de manejar. Al igual que los generadores de eventos, algunas son públicas [19,20,21,22,23] y otras de propósito comercial [24,25,26,27,28], y aunque la mayoría permiten al usuario gestionar el tipo de actividad a monitorizar, también las hay especializadas en áreas.

3. Método para generar eventos

Los programadores que desean hacer pruebas en sistemas IoT, se encuentran los siguientes problemas:

1. No hay una cantidad de eventos suficiente para pruebas.
2. Se necesitan eventos con valores específicos.
3. Hay que esperar la generación de eventos por parte de la fuente de eventos.

Los tres puntos anteriores son claros problemas para los programadores que quieren probar programas que procesan eventos. Por este motivo proponemos un método para generar de forma automática eventos para pruebas. El comportamiento de este método es similar a las plataformas IoT que existen hoy en día, pero con las siguientes diferencias:

1. El usuario puede elegir (sin límite) la cantidad de eventos a obtener.
2. El usuario define el tipo de evento. Se puede personalizar el tipo de evento adaptándose a las necesidades del programador.
3. El usuario puede generar eventos con valores específicos.
4. El usuario puede obtener de forma inmediata los eventos para las pruebas gracias a la automatización del método.

Como según la plataforma IoT que estemos utilizando, el formato de salida de los eventos difiere, se analizan los formatos usados por las siguientes plataformas:

- Buglabs [22] - formato `JSON`
- Carriots [27] - formatos `XML` y `JSON`
- Grovestreams [24] - formato `JSON`
- Lelylan [20] - formato `JSON`
- Particle [21] - formato `JSON`
- Plotly [26] - formatos `JSON` y `CSV`
- Sensorcloud [28] - formato `CSV`
- ThingSpeak [19] - formatos `JSON`, `CSV` y `XML`
- Xively [25] - formato `JSON`
- Zettajs [23] - formato `JSON`

Teniendo en cuenta los formatos de salida existentes en las plataformas IoT, nuestro método generará los eventos en los tres formatos de salida usados por dichas plataformas, es decir, {`JSON, CSV, XML`}.

El método que se propone se divide en etapas (véase la Figura 1) desarrolladas del siguiente modo: el *tipo de evento* del que se desea generar una cantidad de eventos se define siguiendo unas pautas, esta definición será analizada en la etapa de *Validación* y, si esta es correcta, en la última etapa de *Generación de eventos* se obtendrá la cantidad de eventos solicitada, en el formato de salida indicado.

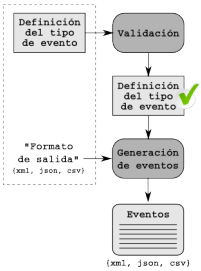


Figura 1. Etapas del método

En las siguientes secciones se detallarán las diferentes etapas del método propuesto para la generación automática de eventos.

3.1. Definición de tipo de evento

El propósito de esta sección es proponer una definición para representar diferentes tipos de eventos. Para intentar uniformar el formato de los eventos, se necesita que se defina un estándar que unifique la definición de los mismos. En [29] Luckham propone las pautas a seguir por un estándar para la representación de eventos:

- Todos los eventos tienen que ser instancia de un tipo de evento.
- La estructura de los eventos se define por el tipo al que pertenece.
- La estructura se representa como una colección de atributos del evento.

En este mismo trabajo se recomienda el uso de un lenguaje de programación fuertemente tipado (como XML Schema o Java), al igual que se indica que cualquier estándar, a la hora de representar eventos, deberá especificar ciertos datos predefinidos (atributos), como por ejemplo:

- Un identificador único para el evento.
- El tipo de evento.
- La marca de tiempo de la creación del evento.
- La fuente de creación del evento.

Además ha de tenerse en cuenta, que los atributos de un evento pueden ser de tipo simple o de tipo complejo.

Siguiendo las pautas y recomendaciones de Luckham, se escoge XML como lenguaje para definir los tipos de eventos. XML es un lenguaje fuertemente tipado del cual existe una gran cantidad de herramientas y librerías que simplifican su proceso de lectura y análisis.

En la Figura 2 se presentan los diferentes componentes, así como sus propiedades, de la definición de tipo de evento que se propone, los cuales se explicarán, utilizando algunos ejemplos, en los siguientes párrafos.

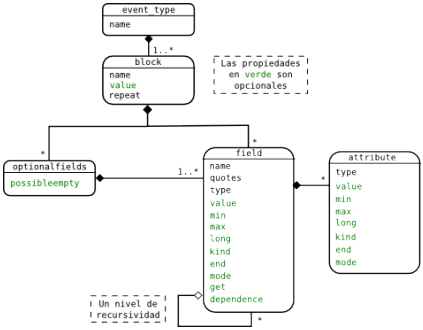


Figura 2. Componentes y propiedades de la definición de tipo de evento.

El componente principal *event_type*, se compone de *blocks* que se identifican con su propiedad obligatoria `name`. Los *blocks* pueden contener tanto *fields* como *optionalfields*. Los *optionalfields* se componen de *fields*. Los *fields* podrán ser de tipo simple o complejo y tendrán asociados unas propiedades. Si es de tipo complejo, se podrá componer de *attribute* o de *field*; este *field* es un tipo anidado y ya no podrá estar compuesto de *field*.

Dado que el lenguaje utilizado para el estándar es XML, se utilizarán etiquetas y propiedades para definir cada uno de los elementos del tipo de evento. Para

que la estructura XML de la definición del tipo de evento pueda ser validada con cualquier validador de este lenguaje de marcado, la primera línea que ha de aparecer es la cabecera con la versión y codificación (véase el Código 1.1). Tras la cabecera, tenemos que indicar el tipo de evento a definir a través de la etiqueta `<event_type>`, donde se indica el tipo de evento a través de la propiedad `name`. En el Código 1.1 se define el tipo de evento `TerminalEvent`.

La etiqueta `<event_type>` se compone de bloques, que se definen con la etiqueta `<block>`. Pueden utilizarse tantos bloques como se quieran, pero uno de ellos tiene que indicar el número de eventos que se desean a través la propiedad `repeat` de la etiqueta `<block>`. Los bloques que no contengan la propiedad `repeat` pueden utilizarse para indicar cualquier tipo de información sobre el tipo de evento. Esta información puede especificarse con la propiedad `value` de la etiqueta `<block>` o a través del componente campo que veremos a continuación (véase el Código 1.1).

Los bloques se componen de campo o campos opcionales, que se definen con las etiquetas `<field>` y `<optionalfields>` respectivamente. Cada campo (`<field>`) define cada uno de los atributos de los que se compone el tipo de evento, y tiene tres propiedades que son obligatorias:

1. Propiedad `name`: contiene el nombre del atributo.
2. Propiedad `type`: se define el tipo del atributo (tipo complejo o simple)
3. Propiedad `quotes`: sus valores son {`true`, `false`}, e indica si los valores del atributo que se define van entrecomillados o no.

En algunos tipos de eventos, ciertos atributos no aparecen siempre; este tipo de atributos se definen con las etiquetas `<optionalfields>`. La única propiedad (opcional) que acepta esta etiqueta es `possibleempty` (valor por defecto `false`), si está a `true` indica que existe la posibilidad de que no aparezca ninguno de esos posibles atributos opcionales, en caso contrario solo aparecerá un atributo. Los componentes de la etiqueta `<optionalfields>` son `<field>` que pueden ser de tipo simple o de tipo complejo. Pueden utilizarse tantos `<optionalfields>` como sean necesarios en la definición del tipo de evento.

En el Código 1.1 se utilizan cada una de las etiquetas y propiedades descritas con anterioridad definiendo el tipo de evento `TerminalEvent`.

Código 1.1. Componentes de la definición del tipo de evento TerminalEvent

```
<?xml version="1.0" encoding="UTF-8"?>
<event_type name="TerminalEvent">
  <block name="head" value="Date:2016;Description:...">
  </block>
  <block name="feeds" repeat="150">
    <field name="status" quotes="true" type="ComplexType"...>
    ...
    </field>
    <field name="terminal_id" quotes="false"
           type="Integer" ...></field>
```

```
    <field name="timestamp" quotes="false" type="Long" ...>
    </field>
    <optionalfields>
      <field name="responsableId" quotes="true"
             type="Integer" ...></field>
      <field name="destinyId" quotes="true"
             type="String" ...></field>
    </optionalfields>
    <optionalfields possibleempty="true">
      <field name="numluggage" quotes="false"
             type="Integer" ...></field>
    </optionalfields>
  </block>
  <block name="location">
    <field name="lat" quotes="false" type="Float"
           value="36.69"></field>
    <field name="lon" quotes="false" type="Float"
           value="-6.11"></field>
  </block>
</event_type>
```

En el Código 1.1, se representa el tipo de evento `TerminalEvent` que se compone de los atributos: `status`, `terminal_id` y `timestamp` y, como atributos opcionales: `responsableId`, `destinyId` y `numluggage`. El elemento `status` es de tipo complejo: `ComplexStatus` y su valor va entrecomillado, y los elementos `terminal_id` y `timestamp` son de tipo simple: `Integer` y `Long` respectivamente, y sus valores no van entrecomillados.

Según la definición del tipo de evento, las posibles estructuras de los eventos de tipo `TerminalEvent` son:

1. {`status`, `terminal_id`, `timestamp`, `responsableId`}
2. {`status`, `terminal_id`, `timestamp`, `destinyId`}
3. {`status`, `terminal_id`, `timestamp`, `responsableId`, `numluggage`}
4. {`status`, `terminal_id`, `timestamp`, `destinyId`, `numluggage`}

Uno de los atributos opcionales (`responsableId`, `destinyId`) del primer `<optionalfields>` tiene que aparecer en el evento generado. Sin embargo, del segundo `<optionalfields>`, existe la posibilidad de que no aparezca el atributo `numluggage` ya que la propiedad `posibleempty` está definida.

Finalmente, los bloques que no contienen la propiedad `repeat`, se han empleado para que en los eventos se incluya información sobre la fecha y descripción del tipo de evento (bloque `head`), así como la localización (bloque `location`).

3.2. Tipos de datos

Siguiendo las recomendaciones de Luckham, hay que tener en cuenta que los atributos de un evento pueden ser de tipo simple o de tipo complejo. En las siguientes secciones se describirán los tipos de atributo simple y complejo, así como las propiedades de cada uno de ellos que hemos contemplado.

Tipos simples

Los tipos simple de nuestra definición son: enteros (`Integer`), de punto flotante (`Float`), enteros grandes (`Long`), cadenas de caracteres (`String`), alfanuméricos (`Alphanumeric`), booleanos (`Boolean`), fecha (`Date`) y tiempo (`Time`). Para definir un tipo simple se le asigna cualquiera de los identificadores anteriores a la propiedad `type` de `<field>`. En el Código 1.2 se muestra un ejemplo de cada uno de los tipos simples contemplados.

Cada tipo simple contiene unas propiedades específicas que determinan no solo el valor del atributo, sino también su formato. Antes de profundizar en cada una de estas propiedades (la mayoría opcionales), hay que indicar que una propiedad común a todos es `value`. Si esta propiedad tiene asignado un valor este es el que tendrá asignado el atributo definido.

Los tipos enteros, de punto flotante y enteros grandes comparten dos propiedades: `min` y `max`, las cuales definen el rango numérico para el valor del atributo (tienen que estar ambas definidas). Si no están definidas, cada tipo tiene un rango de valores por defecto que será el que adquiera el valor del atributo. El tipo punto flotante tiene la propiedad `long`, que indica el número de decimales que tendrá el valor.

Los tipos cadenas de caracteres y alfanuméricos contienen las propiedades: `long`, `kind` y `end`. La propiedad `long` determina la longitud de la cadena (valor por defecto 10). La propiedad `kind` indica si las letras se escriben en mayúscula (valor por defecto) o en minúscula (se indica asignándole a la propiedad el valor `low`). Si se quiere una cadena de caracteres, o alfanumérica hasta un determinado valor, se puede determinar con el atributo `end`.

El tipo booleano tiene una propiedad: `kind`, que si es igual a `n` los valores generados serán {1, 0} y, si no está definida los valores serán {true, false}.

Los tipos fecha y tiempo tienen una única propiedad (obligatoria): `mode`, la cual define el formato y los valores de la fecha o del tiempo.

Código 1.2. Ejemplo de tipos simples y sus propiedades

```
<?xml version="1.0" encoding="UTF-8"?>
<event_type name="TerminalEvent">
  <block name="feeds" repeat="150">
    <field name="terminal_id" quotes="false" type="Integer"
           min="10" max="99"></field>
    <field name="timestamp" quotes="false" type="Long"
           value="2345602942311037"></field>
    <field name="open" quotes="false" type="Boolean" kind="n">
           </field>
    <field name="open_time" quotes="false" type="Time"
           mode="HH:mm"></field>
    <field name="day" quotes="false" type="Date"
           mode="yy-MM-DD"></field>
    <field name="id_plate" quotes="true" type="String"
           kind="low" long="4"</field>
    <field name="hex_color" quotes="true" type="Alphanumeric"
           long="6" end="F"</field>
```

```
    <field name="PoP" quotes="false" type="Float" min="0"
           max="100" long="2"></field>
  </block>
</event_type>
```

En el Código 1.2 se definen los atributos de tipo simple: `terminal_id` (entero), `timestamp` (entero grande), `open` (boolcano), `open_time` (tiempo), `day` (fecha), `id_plate` (cadena de caracteres) and `hex_color` (alfanumérico) and `PoP` (de punto flotante).

El atributo `terminal_id` se representa por cualquier número del rango [10, 99], el atributo `timestamp` tendrá el valor de la propiedad `value`, el atributo `open` es boolcano y tendrá valores numéricos {1, 0}, los atributos `open_time` y `day` tienen los formatos {horas:minutos} y {año-mes-día} (dos dígitos) respectivamente, `id_plate` es una cadena de caracteres en minúsculas cuya longitud es cuatro, `hex_color` es una cadena alfanumérica con longitud seis cuyo tope es la "F" y `PoP` es de punto flotante con dos decimales y un valor en el rango [0, 100].

Tipos complejos

Los tipos complejos se forman combinando cualquiera de los tipos simples y se definen asignándole a la propiedad `type` la palabra clave `ComplexType`. Dentro de los tipos complejos se incluyen los **tipos anidados** con un nivel de anidación, es decir, que un elemento `<field>` podrá tener anidado únicamente otro nivel de elementos `<field>`. En el Código 1.3 se muestran algunos tipos complejos y sus propiedades.

Las propiedades opcionales de los tipos complejos, `get` y `dependence`, afectan al valor del atributo y solamente tienen efecto si están definidas con valor *true*.

Si la propiedad `get` está definida, el comportamiento del tipo complejo a la hora de generar los valores del atributo difiere. En vez de generar valores para todos los hijos del tipo complejo, se escoge aleatoriamente uno de los hijos del tipo complejo para que sea generado.

La definición de la propiedad `dependence` afecta al menos a dos tipos complejos. Si `dependence` se define en un tipo complejo A con valor *true*, significa que al menos un tipo complejo B depende de A. El comportamiento de los tipos dependientes es el siguiente: si del tipo complejo A se escoge el primer hijo, se escogerá el primer hijo de todos los tipos complejos que dependan del tipo complejo A. Dado que la propiedad `dependence` puede aparecer en varios tipos complejos en la definición del tipo de evento, hay que indicar de qué tipo complejo se depende asignándole a la propiedad `dependence` el nombre del atributo definido como tipo complejo. En resumen, si `dependence` es igual a *true*, otros tipos complejos dependen del tipo complejo donde esta asignación aparece, pero si `dependence` es igual al *nombre de un atributo*, quiere decir que este tipo complejo depende del tipo complejo donde se define ese atributo.

Código 1.3. Ejemplo de tipo complejo y sus propiedades

```
<?xml version="1.0" encoding="UTF-8"?>
<event_type name="TerminalEvent">
```

```
  <block name="feeds" repeat="150">
    <field name="terminal_id" quotes="false"
        type="Integer" min="100" max="999"></field>
    <field name="time" quotes="true" type="ComplexType">
     <attribute type="Date" mode="yy-MM-dd"></attribute>
     <attribute type="String" mode="T"></attribute>
     <attribute type="Time" mode="hh:mm"></attribute>
    </field>
    <field name="status" quotes="true" type="ComplexType"
           get="true">
     <attribute type="String" value="OutOfOrder"></attribute>
     <attribute type="String" value="Checkin"></attribute>
     <attribute type="String" value="Cancelled"></attribute>
     <attribute type="String" value="Completed"></attribute>
    </field>
    <field name="departure_loc" quotes="false"
           type="ComplexType" get="true">
     <field name="country_id" quotes="true" type="String"
            long="3"></field>
     <field name="airport_id" quotes="true" type="String"
            long="3"></field>
     </field>
    <field name="destination" quotes="true"
           type="ComplexType" dependence="true">
     <attribute type="String" value="Madrid"></attribute>
     <attribute type="String" value="Rome"></attribute>
     <attribute type="String" value="Paris"></attribute>
     <attribute type="String" value="Berlin"></attribute>
    </field>
    <field name="acronym" quotes="true" type="ComplexType"
           dependence="destination">
     <attribute type="String" value="MDR"></attribute>
     <attribute type="String" value="RME"></attribute>
     <attribute type="String" value="PRS"></attribute>
     <attribute type="String" value="BRL"></attribute>
    </field>
    </block>
</event_type>
```

En el Código 1.3, el tipo complejo *time* se compone de varios tipos simples, un posible valor generado: "16-07-12T12:30". El tipo anidado *departure_loc* contiene los tipos simples: *country_id* y *airport_id*, definidos como campos (con sus correspondientes propiedades). En los tipos complejos *status* y *departure_loc*, se emplea la propiedad `get`, para ambos se escogerá alcatoriamente uno de los hijos para generarse. El atributo *status* tendrá uno de los siguientes valores: {`OutOfOrder, Checkin, Cancelled, Completed`}, y para *departure_loc* se generará un valor para `country_id` o para `airport_id`. Finalmente, el atributo *acronym* depende de *destination*, si se escoge el tercer hijo de *destination*, (`destination:Paris`), se escogerá el tercer hijo de *acronym*, (`acronym:PRS`).

3.3. Etapa de validación

Una vez definido el tipo de evento, se procede a la etapa de *Validación*, en la que se comprueba si la definición del tipo de evento está bien construida de acuerdo a la descripción expuesta en la sección anterior.

En la validación se hace un recorrido por todas las etiquetas que componen la definición del tipo de evento. Se comienza por la etiqueta raíz `<event_type>` comprobando que esta solo contenga etiquetas `<block>` como hijos, que todas contengan la propiedad `name` y que la propiedad `repeat` aparezca una vez.

A continuación comprobaremos los hijos de la etiqueta `<block>` que contiene la propiedad `repeat`; estos tienen que ser `<field>` u `<optionalfields>`. Para la validación de los hijos `<field>` se comprueba que contengan las propiedades obligatorias: `name`, `quotes` y `type`. Para validar los `<optionalfields>`, únicamente se controla que sus hijos sean `<field>`, junto con las anteriores comprobaciones.

Una vez comprobada la estructura del tipo de evento, se validan los tipos de datos: simples y complejos. Para los tipos simples se comprueban las propiedades obligatorias de cada uno de ellos: `name`, `quotes` y `type` para los tipos simples definidos con `<field>` (ya comprobado), y `type` para los tipos simples definidos con `<attribute>`. Además, según el tipo simple (enteros, de punto flotante, enteros grandes, cadenas de caracteres, alfanuméricos, booleanos, fecha y tiempo), controlaremos si las propiedades optativas están bien definidas. Para los tipos complejos se verifica que sus hijos son `<field>` o `<attribute>`, y se validan los tipos de datos simples que lo componen.

Si alguna comprobación anterior no se cumple, se advierte del error.

3.4. Etapa de generación de eventos

El propósito de esta etapa, es generar eventos de manera automática. En esta última etapa de nuestro método se trabaja con el fichero XML validado con la definición del tipo de evento y el formato de salida de los eventos: `JSON`, `CSV` o `XML`. Hay que tener en cuenta que la estructura de los eventos generados variará según el formato de salida indicado, si el formato escogido es `XML` aparecerán `<etiquetas>`, si se desea en formato `JSON` aparecerán {llaves}, o ',' si el formato escogido es `CSV`. La estructura de los eventos generados es algo que se tiene en cuenta a lo largo de todo el proceso de generación.

En esta etapa se sigue un proceso similar a la etapa de *Validación* ya que se comienza por la etiqueta raíz `<event_type>` leyendo cada uno de los hijos `<block>`. Cuando nos encontramos la propiedad `repeat` se profundiza en la etiqueta `<block>` que la contiene leyendo sus hijos `<field>` y/o `<optionalfields>`. Si la definición de un atributo es de tipo simple, se leen las propiedades específicas de dicho tipo y se generará un valor aleatorio cumpliendo las restricciones de las propiedades. En caso de no tener propiedades específicas, el valor generado aleatorio cumplirá las propiedades por defecto del tipo. Si la definición del atributo es de tipo complejo, si están definidas las propiedades `get` o `dependence`, actuaremos en consecuencia. De forma aleatoria cogeremos uno de los hijos del

tipo complejo que tenga `get`, y se generará su valor según las propiedades específicas del tipo de dato simple que lo defina. Del mismo modo (aleatoriamente) cogeremos uno de los hijos del tipo complejo que tenga `dependence`, y el de la misma posición de los que dependan de él.

4. Casos de prueba

ThingSpeak [19] es una plataforma de datos abierta y una API para el IoT, que permite a cualquier usuario recoger, almacenar, analizar, visualizar y actuar sobre datos (en forma de eventos) que provienen de sensores o transmisores como Arduino, Raspberry Pi, BeagleBone Black y otro tipo de hardware. Esta plataforma contiene una serie de canales en los cuales existe la posibilidad de compartir dicha información haciéndolos públicos.

Cada día esta plataforma crece, ya que son muchos los usuarios que quieren compartir sus datos a través de estos canales (más de 5000). En cada uno de esos canales se define un tipo de evento (la estructura), que determina el tipo de estudio que se está realizando. Los tipos de evento definidos en estos canales son para estudios meteorológicos, térmicos, referentes a la luz, agua, humedad, ruido, fermentación, conexiones de Internet, etc. Para este trabajo se han analizado los tipos de eventos de más de 450 canales, aproximadamente un 10 % de los canales disponibles en el momento del estudio (como se ha comentado el incremento de canales de la plataforma es considerable). El listado de los canales analizados de la plataforma "ThingSpeak" se encuentra localizado en el repositorio de UCASE[1].

En la Tabla 1 se recogen algunos canales de ThingSpeak estudiados, los eventos que lanza el correspondiente canal en uno de los formatos {JSON, XML, CSV}, su definición en XML y los eventos generados a través de nuestro método.

Canal	Eventos en ThingSpeak	Definición	Generados
11467	Formato JSON	Office Environment	100
21362	Formato JSON	Tiraspol, MD	500
3	Formato JSON	ioBridge Server	1000
41393	Formato XML	IOT House - Basement	100
5186	Formato XML	Channel 5186	500
65409	Formato XML	Pella Weather	1000
77777	Formato CSV	ArduinoProject	100
8557	Formato CSV	Fibaro HC2 - Temperatura	500
9960	Formato CSV	Raspberry Pi - Apartment	1000
1417	Formato JSON	CheerLights	250

Tabla 1. Tipos de eventos de ThingSpeak generados a través del método

Cada uno de los datos que aparecen en la Tabla 1, son enlaces que nos llevan a la información sobre el canal en sí (Canal), el conjunto de eventos

[1] `https://neptuno.uca.es/redmine/projects/sources-fm/repository/show/trunk/src/IoT-EG/test/ChannelsThingspeak`

lanzado actualmente por el canal (Eventos en ThingSpeak), el fichero XML con la definición del tipo de evento (Definición) y el conjunto de eventos generados a través de nuestro método, cantidad especificada en la tabla (Generados).

Para la generación de eventos en los diferentes formatos se escogen: 4 en JSON, 3 en XML y 3 en CSV. Los resultados muestran que no hay diferencias entre los eventos generados y los de la plataforma. Se demuestra que el método propuesto cubre una gran diversidad de tipos de eventos, lo cual va a permitir que cualquier usuario genere la cantidad de eventos que desee, de forma automática y personalizada, (con valores específicos o aleatorios) para sus pruebas.

5. Conclusiones y Trabajo futuro

Hacer pruebas en sistemas IoT, los cuales consumen y procesan grandes flujos de eventos, es un reto. Consideramos fundamental analizar estos sistemas a través de los eventos que procesan para estudiar su comportamiento en situaciones críticas. Desafortunadamente las investigaciones existentes no están enfocadas en esta dirección. Hoy en día los programadores se pueden encontrar problemas para probar estos sistemas: falta de eventos, valores específicos en los eventos, espera de generación de eventos por parte de la fuente. En este trabajo se propone un método que solventa los problemas anteriores cuyas etapas son: definir el tipo de evento, validar la definición del tipo de evento y generar los eventos según la definición validada.

Este método ha sido probado con más de 450 tipos de eventos reales alojados en la plataforma ThingSpeak, que han servido no solo para validar la funcionalidad del método propuesto, sino también para precisar la definición de tipo de evento en las primeras fases.

Dada la cantidad de flujos de eventos que consumen los sistemas IoT, este método se presenta como una buena alternativa para testearlos, ya que se pueden generar tantos eventos como se deseen, y en los formatos más comunes en los que estos se presentan. Además, gracias a la flexibilidad de la especificación presentada, se pueden definir una vasta variedad de tipos de eventos (con valores aleatorios, dentro de un rango o específicos) para realizar diferentes pruebas en este tipo de sistemas; desde pruebas negativas (se establece el rango de los valores inválidos en la definición del tipo de evento), a prueba de mutaciones (se generarían eventos con valores específicos para detectar los errores cometidos).

Nuestro trabajo futuro se centra en la generación de valores más específicos para los eventos que permita mejorar la calidad de los casos de prueba. Entre otras mejoras se está implementado una funcionalidad para generar eventos con valores que se obtengan de las consultas EPL que van a manejar dichos eventos. Por otro lado, se plantea la posibilidad de generar valores secuencias en algunos tipos simples de modo que se permita una secuenciación en dichos atributos.

Referencias

1. Haller, S., Karnouskos, S., Schroth, C.: The internet of things in an enterprise context. Springer (2008)

2. Luckham, D.: The power of events. Volume 204. Addison-Wesley Reading (2002)
3. Schiefer, J., Rozsnyai, S., Rauscher, C., Saurer, G.: Event-driven rules for sensing and responding to business situations. In: Proceedings of the 2007 inaugural international conference on Distributed event-based systems, ACM (2007) 198–205
4. Dunkel, J., Fernández, A., Ortiz, R., Ossowski, S.: Injecting semantics into event-driven architectures. In: ICEIS (1). (2009) 70–75
5. Luckham, D., Schulte, R.: Event processing technical society. Event Processing Glossary—Version **2** (2011)
6. Habibi, E., Mirian-Hosseinabadi, S.H.: Event-driven web application testing based on model-based mutation testing. Information and Software Technology **67** (2015) 159–179
7. Weiss, J., Mandl, P., Schill, A.: Introducing the qcep-testing system for executable acceptance test driven development of complex event processing applications. In: Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation, ACM (2013) 13–18
8. Mendes, M., Bizarro, P., Marques, P.: A framework for performance evaluation of complex event processing systems. In: Proceedings of the second international conference on Distributed event-based systems, ACM (2008) 313–316
9. Mendes, M.R., Bizarro, P., Marques, P.: A performance study of event processing systems. In: Performance Evaluation and Benchmarking. Springer (2009) 221–236
10. Li, C., Berry, R.: Cepben: a benchmark for complex event processing systems. In: Performance Characterization and Benchmarking. Springer (2013) 125–142
11. Li, C.: Performance management of event processing systems. PhD thesis, Aston University (2014)
12. EsperTech: Espertech website. `http://www.espertech.com/esper/index.php`
13. Oy, M.R.F.: Event generator. `http://www.mrf.fi/index.php/timing-system/71-event-generator`
14. Systems, S.: The event generator. `https://www.starcomsystems.com/ru/the-event-generator`
15. Oracle: Introducing the weblogic integration administration console. `https://docs.oracle.com/cd/E13214_01/wli/docs85/manage/intro.html`
16. Dobbs, M.A., Frixione, S., Laenen, E., Tollefson, K., Baer, H., Boos, E., Cox, B., Engel, R., Giele, W., Huston, J., et al.: Les houches guidebook to monte carlo generators for hadron collider physics. arXiv preprint hep-ph/0403045 (2004)
17. Chekanov, S.: Hepsim: a repository with predictions for high-energy physics experiments. Advances in High Energy Physics **2015** (2015)
18. Mangano, M.L., Stelzer, T.J.: Tools for the simulation of hard hadronic collisions. Annu. Rev. Nucl. Part. Sci. **55** (2005) 555–588
19. ThingSpeak: Thingspeak website. `https://thingspeak.com`
20. Lelylan: Lelylan website. `http://www.lelylan.com/`
21. Particle: Particle website. `https://www.particle.io/`
22. Buglabs: Buglabs website. `http://buglabs.net/bugswarm`
23. Zettajs: Zettajs website. `http://www.zettajs.org/`
24. Grovestreams: Grovestreams website. `https://grovestreams.com/index.html`
25. Xively: Xively website. `http://xively.com/`
26. Plotly: Plotly website. `https://plot.ly/`
27. Carriots: Carriots website. `https://www.carriots.com/`
28. Sensorcloud: Sensorcloud website. `http://sensorcloud.com/`
29. Luckham, D.C.: Event processing for business : organizing the real-time enterprise. Hoboken, N.J. John Wiley & Sons (2012)

A Software Engineering Experiments to value MDE in testing. Learning Lessons

M.J. Escalona[(1)], G. Lopez[(1,2)], S.Vegas[(3)], L. García-Borgoñon[(1,2)], J. A. García-García[(1)], N. Juristo[(3)]

[(1)]Grupo IWT2. Universidad de Sevilla
[(2)]Instituto Tecnológico de Aragón
[(3)]Universidad Politécnica de Madrid
mjescalona@us.es, glopez@itainnova.es, svega@upm.es, laurag@itannova.es, julian.garcia@iwt2.org, juristo@upm.es

ABSTRACT
Controlled experiments are commonly used to evaluate Software Engineering methods, processes and tools. Validating results of Software Engineering research in industrial settings is a direct way to obtain feedback about its value. However, few software engineering experiments are running in industry. The lack of communication between companies and research teams does not make the necessary cooperation among them possible. This paper presents our experiences when running an experiment dealing with Early Testing at the University of Seville. It also introduces the strategy we followed to obtain the participation of 97 practitioners from 32 different software companies. Such strategy is pointed out as a set of guidelines to successfully involve this large number of companies and practitioners.

Keywords: Software experiments in enterprise, early testing experiences

1 INTRODUCTION

Empirical studies are becoming increasingly common in Software Engineering and the acceptance that their contributions are enriching the knowledge of this area is continuously increasing. Empirical studies are needed to develop or improve processes, methods and tools for software development and maintenance [1].

Controlled experiment is a type of empirical study. It allows the identification of cause-effect relationships [24]. The experimental paradigm proposes that laboratory findings should be generalized through other types of experiments closer to real world. In Software Engineering, the equivalent to field experiments is experimentation in industry [26].

A major ambition of experimental Software Engineering is to provide software managers, who are in charge of decision-making processes in software development industry, with evidence on how new technology can be introduced [16]. However, the experimental subjects of these experiments are typically studied with little or no professional experience [3]. The few experiments carried out with practitioners are isolated experiences [26]. Additionally, the experimental setting and materials tend to be artificial or only partially related to real projects [24].

The systematic use of experiments as a way to face the decision analysis and resolution practices is far from what would be desirable.

In [26], authors report preliminary results regarding the difficulties encountered when they run experiments in the industry. One of their findings is that reporting used in scientific journals is not appropriate for practitioners and what is more important, frequently in reported experiences the number of participants is very low and research community has real problems to evolve a good number of them in experiments.

The goal of this paper is to present a set of guidelines (good practices) for involving a large number of experimental subjects and companies when running experiments with practitioners. To illustrate these guidelines, we present a real experiment conducted at the University of Seville, with a total of 97 practitioners registered (76 of them finally participating in the experiment) from 32 different software companies to evaluate our research results in Early Testing.

The reminder of this paper is organized as follows: Section 2 presents both, a general view of software experiment and the motivation for our experiment. Section 3 describes our experiment in detail, and Section 4 numbers and explains the mechanisms and strategies followed to get an effective participation of companies. Section 5 summarizes learned lessons and finally, the paper ends up by stating conclusions and future lines of work in Section 6. It is important to stick out that this paper does not aim to present the results of our experiment, but to recommend a strategy that can help research teams to involve a large number of practitioners and companies in software engineering experiments.

2 BACKGROUND

Dieste *et al*. [3] report the preliminary results of a systematic literature review exploring the features of experiments run in industry. They have located a total of 15 studies. The results highlight that the software engineering community perceives experimenting in industry as a problematic activity: ***few experiments have been run and companies are demotivated to carry them out since they do not realize their value and direct benefits***. Jedlitschka *et al.* [15] empirically develop a model that characterizes software engineering practitioners information needs. They evaluate the effectiveness of the proposal with 22 software managers. ***The results have shown that experiments can be a valuable source of information in industry environments, although they do not solve the problem of involving companies in conducting them.***

Vegas *et al.* [26] discuss the difficulties they identify when running experiments in software industry. One of their main finding is the difficulty to involve a large number of participants from industry. In view of these results, it seems that experiments could be valuable for the software industry. However, few experiments are run in industry, and they do have a small number of participants. Therefore, it is essential to find a way to encourage software engineering companies and practitioners to participate in experiments. We planned an experiment taking advantage of the relationship some of us have with some software development companies. The Web Engineering and Early Testing (IWT2) research group [27] has an extensive experience in technology transfer and has set a very fruitful relation with companies through their joint collaboration in a large number of software projects [6]. Thus, we conducted an experiment with a main objective: to validate our research in Early Testing. Besides, we wanted to value some strategies in order to make practitioners participate in that project. In fact, this paper mainly focuses on this secondary objective. For that purpose, we present the strategy we have followed to successfully enrol a large number of industrial participants in our experiment.

3 DESCRIPTION OF THE EXPERIMENT

3.1 Experiment Definition

The goal of the experiment is to ***analyze the adequacy of the paradigm driven by models (or MDE-Model Driven Engineering) [25] as the basic technique for developing functional tests in an early phase (Early Testing)***. The experiment was opened to all software engineers who do some kind of testing in their daily work. Initially, 97 participants registered in the project, although some of them were not able to assist. Finally, 76 practitioners from 32 different software companies participated in the experiment. The call for the experiment was opened to staff of companies (small, medium and large) and software organizations (private and public) based anywhere in Spain. Participants mainly came from Andalusia, but people from other areas like Madrid or Barcelona also took part in the project. They were not required a particular profile, apart from being a software engineer who performs some kind of software testing in their daily work. Thus, juniors and seniors testers, test managers or developers who do testing in a timely manner were able to participate. That heterogeneous group required the collection of demographic data of each participant in order to guarantee a proper randomization that gave us a right profile of each participant. For this purpose, an electronic form was used for registration[1]. The contents of this form are summarized in Table 1.

Table 1. Data gathered in the registration form.

Personal Data	• Name • Surname • NIF[2] • Date of Birth • Telephone number • Email address • Gender
Academic Data	• Degree • Date of degree • Any special course related to testing (such as TestQA)
Professional Data	• Company • Position: − Manager − Analyst − Designer − Programmer − Scholar employer − Other • Years of experience in testing or in other fields (like programming or commercial activities, among others) − Experience in testing: − Test manager

[1] The questionnaire was designed in Spanish and it was developed using Opina[19]. It is available in http://iwt2.org/opina/c/290

[2] NIF (Número de Identificación Fiscal) is a personal number that each person is assigned in Spain by the Government.

	– Junior tester – Senior tester • Percentage of time spent in testing per week: – Less than 25% – Between 25% and 50% – Between 50% and 75% – More than 75% • Typical testing performed in his/her work: • Unit testing • Integration testing • Functional testing • System integration • Acceptance testing

There are two research questions in the experiment:

- **RQ1:** Is the developer effort affected by MDE when generating functional test cases?
- **RQ2:** Is the quality of the functional test cases generated affected by MDE?

Each of these research questions has an associated hypothesis (H):

- **H01**: The effort to build a test case using MDE is the same as using manual development.
- **H02**: The control and quality of test cases generated using MDE is the same as using manual development.

The experiment includes one relevant factor: functional definition method. Control is a manual method while treatment is a MDE method supported by an NDT-Suite [10] solution. NDT-Suite consists of a set of tools to support NDT (Navigational Development Tecniques)[5][7]. It is not the aim of this paper to present NDT in detail, nevertheless, we can briefly introduce that it is a model-driven methodology that offers suitable support for the application of MDE in software development. NDT covers the whole software lifecycle, from requirements to maintenance, and it even supports other management activities like quality assurance, project management and software security assurance. NDT has been successful applied to a large number of real projects, even though its suitability for companies was never methodologically valued at companies. As NDT covers many tools and software phases, we selected the most interesting ones for our first experimentation with companies. Besides, it must be added that one of the most important characteristics of NDT is the support it provides for Early Testing activities. Testing is currently a critical phase for software companies because they usually have poor resources [11]. Thus, we selected concretely the functional phase. The experiment aimed to test whether, in fact, experts consider that these solutions are suitable enough for enterprise environments.

The response variables measured in the experiment are:

- RQ1 required measuring the developer effort. The selected metric was the ***resolution time***.
- RQ2 was a little more complicated. Initially, as our experimental objects came from real projects (as explained in section 3.2.1), we had the code of the system in the first version (and also in the final one). Thus, if experts detected a good number of errors or mistakes, we considered that they defined a good set of functional tests. However, it was not possible. Although this aspect will be treated in section 4, tests defined manually by our experts did not detect any errors (the reason will be analyzed later), so we reconsidered this way of evaluation and we decided that two experts would assess the functional test definition, would value the participant tests and would give them a mark that ranged from 0 (very bad) to 4 (very good).

3.2 Experimental Design

The experiment follows one factor-two level (control and treatment) within subjects design.

Table 2 presents the experimental operation. The experiment had two differentiated parts. In the first part, participants were asked to exercise the control. This implied to generate functional tests manually. We did not offer any support or extra learning because we aimed to assess the knowledge of our experts in that field. In the second part, after a training session, we asked participants to generate functional tests using MDE with NDT. After that, each participant filled out a form.

We took into account several points before defining the schedule:

1. We had to offer, at least, two different software systems; one manual and other automatic. Thus, we first selected PIF (Information Flamenco Point)[21] and Ambassador Hotel [12] projects. Nonetheless, there was a first idea learned for that selection. When we designed our experiment, we considered that the systems should be easy to understand by our participants and they should be real systems to be more attractive for them. Thus, we were thinking about other true projects in which our group collaborated like aeronautical projects [8] or health projects [18]. Finally, we discarded them because their functionality required a previous training session on the software application domain, and we preferred to involve a big group of different practitioners. That was the right decision, as the chosen systems did not require extra time to explain our use cases to participants.
2. The complexity of the system was also another important decision. We selected a set of use cases from each of our systems; two easy use cases (one from each system) and two difficult ones (also one from each system). Thus, we had four different tasks for our experiment:
    a. Ambassador Easy, we coded as D1F
    b. Ambassador Difficult, we coded as DID
    c. PIF Easy, we coded as D2F
    d. PIF Difficult, we coded as D2D

Table 2. Timetable for the experiment[3].

Time	
15:30-16:00	Welcome and presentation
16:00-16:30	Presentation of examples and material
16:30-17:00	Defining Functional Testing manually
17:00-17:30	Coffee breack
17:30-19:00	Training course: knowing MDE. Presenting NDT and NDT-Suite
19:00-19:30	Defining Functional Testing automatically
19:30-19:45	Final interview
19:45-20:00	Closing and final conclusions

According to the experiment definition presented in Table 2, each participant must perform two tasks, one manually, and one automatically using MDE, concretely, NDT and NDT-Suite as Table 3 shows.

[3] This table presents an afternoon section. Further information can be found in the website of the experiment: http://iwt2.org/experimentacion-en-testing-temprano/.

.

Table 3. Experimental design.

	Manual	**Automatic**
Group 1	D1F	D2F
Group 2	D2F	D1F
Group 3	D1D	D2D
Group 4	D2D	D1D

Subjects were assigned to groups using stratified randomization. For that purpose, we used the data collected in the registration form. Table 4 shows the results of the randomization.

Table 4. Distribution of experts for experiments.

	D 1F	D1D		D2F		D2D		Total	
	M	A	M	A	M	A	M	A	
Manager	2	2	1	5	2	2	5	1	12
Analyst	7	2	4	4	2	7	4	4	17
Designer	4	3	3	4	3	3	4	4	15
Programmer	3	4	2	3	4	3	3	2	12
Scholar employer	0	0	1	0	0	0	0	1	1
Other	5	8	12	5	8	5	5	12	24
Total	21	19	23	21	19	20	21	24	97

During the experiment, we prepared a personal folder for each participant that included the use case diagrams of the task that he/she had to perform manually. In this case, participants also received an activity diagram that described one of the use cases of the diagram in detail. Besides, they were given a set of templates (4 concretely) for functional tests. Everyone got only the activity diagram that they had to execute, and they did not know what their colleagues were doing. In the first part of the experiment, each participant had to define up to four functional tests manually only with the information of the use case and the activity diagram, but using the test functional pattern included in the folder, as we previously mentioned. We concretely defined the maximal number of functional tests because in real projects, companies do not have as many resources as they would like. They have a limited one and we restricted them to four using the real experience of selected projects in the practice. The detailed description of use cases and activity diagrams is out of the scope of this paper. However, further information on the experiment can be found in our group's website.[4] After the manual exercise, our experts taught a course related to Early Testing. In that course, participants were given a detail description on the advantages of Early Testing, reduction of time and how it can help the early detection of errors in software lifecycles. During the course, participants discussed with our experts and worked with tools they would use in the second part of the experiment. After the course, our participants received a new document, the second use case and the activity diagram (different from the previous one) to be executed with NDT-Suite. Along that execution they received the functional test derived from their activity diagram only with a click. Finally, once the experiment finished, we asked them to talk about their experiences in Early Testing and its automation. This form is available in [5].

[4] http://iwt2.org/experimentacion-en-testing-temprano/

[5] This report was also presented in Opina. Obviously, obtained results were really interesting.

3.3 Running the Experiment

The right execution of the experiment was a critical aspect for our future relationship with the participating companies. For this reason, we ran a pilot in Zaragoza before the experiment. We counted with the support of the Instituto Tecnológico de Aragón[14], which invited 17 companies. During this experience, we were able to assure that our schedule was well planned, that some aspects of the Early testing course had to be changed to make them clearer for participants and that some other improvements were necessary. Additionally, three days before the experiment, we ran a new pilot in the same classroom where the real task would be conducted, with five students in the last course of the degree. These two pilots would ensure that any technical problem would be solved before the experiment. A total of 91 professionals registered for the experiment. Some of them were people who had previously worked with us in other projects (they could be consider our "costumers" in Technology Transfer projects) and some others were going to collaborate for the first time in a University project. The experiment was carried out on 4th and 5th. It was planned as a half-day activity (4.5 hours, see Table 2) developed at the School of Computer Science at the University of Seville. At the beginning of the session, we asked them to fill in a registration form that included the collection of personal data and academic and professional information similar to that in Table 1. It would guarantee that nobody made an error in his/her inscription. During the execution, we had two problems:

1. Although we had 91 people registered, only 76 participants took place in the experiment. Therefore, to mitigate this problem, the remaining people were re-assigned to groups in order to maintain the original design balanced.
2. The second problem was related to the number of computers but, it will be analyzed in detail in the next section.

As commented, information, both inscriptions and interviews at the end of the experiment was gathered through online forms, designed with OPINA; a free tool frequently used because it is very versatile and lets us explore and export results in Excel sheets. The first part of the experiment, that was to say, the manual definition of tests, was made by hand in paper. We compiled and classified them according to the starting and ending time of each participant (key for measure the first RQ). As previously mentioned, participants used NDT-Suite for the second part of the experiment consisting in the generation of tests automatically. This tool can be briefly described as a specific UML (Unified Model Lenguages) [20] profile that makes possible to apply NDT to real environments. NDT-Suite also implements all necessary transformations among models. All this framework is integrated into Enterprise Architect [4] tool. This environment, allowed each participant to be able to (i) generate tests associated with his/her use cases and (ii) produce different office documents with the structure of each test and results. That process was performed in an automatic and systematic way, but we kept the results of each participant performance to control that part of the experiment. We also recorded the time each of them spent (i.e., starting and ending time).

3.4 Threats to Validation

Our design was exposed to several validity threats such as fatigue and learning. We tried to address them by applying the following strategies:

- **Fatigue effect:** The fact that each subject has to analyze two problems means that two sessions are required to run the experiment. If these sessions are performed in close succession, subjects may experience fatigue, and in turn, they may lose effectiveness as the experimental sessions progress. To avoid this pernicious effect, the experimental sessions will take only 4.5 hours, with a coffee break to relax. Therefore, fatigue will not affect the second session, which will be developed in similar conditions to the first one.

- **Learning effect:** The source of the learning effect is the performance of the same experimental task by the same experimental subject on repeated occasions. In this experiment, each subject will study two completely different domains, and consequently the information will be unlikely to reuse from the domain-aware problem (AP1) to domain-ignorant problem (IP1). Besides, the elicitation is performed using the open interview, and subject skills are a priori unlikely to improve substantially after a mere 30-minute course (actual elicitation sessions were even shorter) and over the 2 days between sessions.

4 PRACTICAL EXPERIENCE WORKING WITH COMPANIES

This section explains how the strategy illustrated in section 3 has been embodied in a concrete experiment to get a large number of participants. It has been organized around three moments: before the running the experiment, along the experiment and after running the experiment.

4.1 Before Running the Experiment

The first idea was ***to attract our enterprise experts through their clients***. Thus, we contacted with Mr Jesús Huertas, Director General de Política Digital in Consejería de Hacienda y Administración Pública in Junta de Andalucía[2], who is responsible for the Andalusian Government IT (Information Technologies) digital policies. He is a very relevant professional, as well as a key point of contact or an important client for companies in Andalusia. The manager of our team asked for a personal meeting with him, as she knew him for several years, she presented him the experiment and asked for his endorsement. She explained in detail the importance of this kind of experience carried out among University-Public Administration-Companies, mainly in a current strategy line for IT: testing (it took her 30 minutes approximately). The endorsement that we were looking for was critical for our experiment. That meeting enables her to explain that we were also backed up by Junta de Andalucía, which had recognized our experiment as a very relevant initiative for the community. After that, as a second action ***we prepared an invitation and we designed a website with the definition of our experiment***. In that invitation, we explained some critical aspects[6]:

1. The support of Junta de Andalucía and Mr Jesús Huertas' backup.
2. A set of advantages for our participants, described below.
3. A very concrete agenda.

These aspects are themselves an attractive claim for companies for several reasons:

1. The support of Junta de Andalucía was essential because they have key clients for companies in our environment and they also set the pace and make global decisions about IT that are decisive for IT companies in Andalusia. Thus, to collaborate with the University of Seville in a Software Testing project endorsed by Mr Jesús Huertas gives companies a relevant position in the community.
2. The advantages we presented were also fundamental, as we offered two different points.
   a. First, advantages for participants. Thus, we offered a personal certificate of participation, a free course on Early Testing and a free evaluation of their performance along

[6] Once again, it is available in the website of the experiment http://iwt2.org/experimentacion-en-testing-temprano/

the experiment, where we valued their knowledge. Besides, a personal evaluation from experts in our team is also offered.

b. Second, advantages for companies. Thus, if a company participated at least with five experts, it would receive an especial certificate of participation. That certificate resulted very attractive for them, mainly for future public contracts in Andalusia.

3. Defining a very concrete agenda was also a relevant aspect. People in companies are frequently very busy; to spend time in an experiment is often very difficult for them, so an agenda may help them to close and reserve a specific period of time for the experiment. Besides, we set three different sessions, with the idea that they could adapt their participation in the project to their professional agendas.

However, our experience with companies lets us know that an official invitation is not enough, as everyday they receive a lot of emails and some of them, despite their attractive designs, are usually classified as spams. Consequently, if we only sent the invitation, it would probably be discarded and considered as a spam. Thus, with the official invitation and as a third strategy, ***we sent personal emails to companies' staff***. The fact of sending the invitation helped us to learn that the two points below were very important:

1. To send individual emails with a personalized text for each recipient. We did not send our mails to companies in general. They were sent to concrete people (our own contacts, our contacts' contacts or even recommended addressees by Junta de Andalucía). Each email was different and each of them attached the personal invitation. where we asked participants to resend it to other colleagues.
2. To contact people through phone calls, WhatsApps, social nets or other similar platforms of communication, as we considered that, despite this personal mailing, people quick rhythm of life would make them discard the email without reading it..

Obviously, those kinds of calls were very expensive in resources. In our experiment, it was the director of the team who mainly executed that task, which was also a very relevant aspect; for companies, if it is the boss who makes the phone call, the fact will acquire more relevance.

Table 5 below numbers the mails, WhatsApps and phone calls made for inviting people.

Table 5. The number of contacts made by the IWT2 leader during the call is presented [7] .

Number of sent mails	337
Number of direct phone calls related with the experiment	131
Number of sent WhatsApps directly related with the experiment	350[8]

However, we considered that this personalized dedication was not enough. It was key for our experiment to attract different profiles from SMEs (Small and medium enterprise), big companies and public companies. Besides, we claimed for different levels of experts: senior and junior people. Thus, we also asked for a face-to-face interview in case of key participants. Consequently, as a fourth strategy, we demanded face-to-face meetings with some companies' members of the staff who were very important for our experiment. Thus, for instance, with ATOS[1], whose international testing factory is based in Seville, or with IECISA[13][9], we held a personal meeting with the direction of the factory to invite them to participate in our project.

[7] This table only presents specific calls and mails for this experiment. In other meetings or environments, the experiment was also referred but it was not presented in this paper.

[8] It is a reference since a conversation can include several messages. It refers to the number of conversations.

[9] IECISA: Informática de El Corte Inglés

Table 6 offers the number of meetings that the leader of our team held with different companies, grouped into SMEs, big companies or public companies. We deemed to have such meetings in the companies' headquarters as something relevant, despite the cost of displacement,, as some of them took place even out of Seville.

Table 6. The number of face-to-face meetings held by the IWT2 leader during the presentation of the call.

SME visits	Public companies visits	Big companies visits	Total
4	4	5	13

We can assure that all visited companies participated in the experiment. Some of them even asked our team to replicate the experiment in the own company. Thus, as we will conclude in the last section, one-to-one meetings were very expensive, but a very good and profitable strategy.

Table 7. Total number of participant.

	Senior	Medium	Junior	Testing Experts	Gender		**Total**
					men	women	
SME	2	12	2	1	15	1	**16**
Big Companies	26	15	14	34	45	10	**55**
Public companies	8	5	3	0	12	4	**16**

Finally, to finish with our experiment arrangements, we defined an online registration form described in Table 1. ***The strategy of having an online registration form was also a suitable idea.*** That form would provide us with relevant aspects as follows:

1. We would follow the number of participants and the influence of our calls. Thus, if a company or someone was not inscribed, we would insist with more calls or mails.
2. We included some demographical and personal data (like experience or level of test knowledge, for instance) that would help us to define the scope of our experiment, as it will be further explained in the next subsection.
3. Each participant would select the best section for him/her. That was critical to assure that we would assume the number of participants in each section.

After carrying out the task of contacting people, Table 7 shows the final number of participants.

4.2 Along the experiment

Experiments were executed on November 4th and 5th in three different sessions; one on Wednesday afternoon and the other two on Thursday morning and afternoon, respetively. We did not define the proceeding of those sessions, as in fact they were defined in collaboration with companies, particularly with bosses along the face-to-face meetings previously held with them. ***Coordination with companies*** was a successful idea in order to guarantee the participation of our experts. Another important issue was the preparation of the class. As we commented, we executed a similar project in Zaragoza some weeks before. Besides, on Monday before the agreed date for the experiment, seven people executed it in the same classroom where it would take place, just to check if the software was right, if computers were available, if web connection was running properly, and some other

concerning aspects. ***This test was considered as a very interesting strategy since it gave us time to repair some computers with software problems*** (just technical problems when installing Enterprise Architect or with the internet connection). It is also remarkable that some days before the experiment, we ***sent personal emails to each participant as a reminder of the timetable, the agenda and the some other arrangements related to his/her participation in the project***. Those emails aimed not only to remind, but also to confirm their attendance to the meeting.

As we introduced, we worked with two different systems and with two kinds of use cases for each of them: four examples in total. Then, we divided cases with on-line supporting among our participants, in order to assure their good distribution. The email enabled us to guarantee that we had a right number of examples and a right distribution. However, we also considered other elements, not only based on the methodology for software experiments, but also related to the corporative and public image of our group and our experiment. They were mainly these three listed as follows:

1. Each participant received a folder with personalized documentation for the experiment. In fact, we were sponsored by Fidetia[9], which provided folders, pens and notebooks, by offering a very serious and interesting view of our experiment.
2. The coffee break was seen as a relevant moment that allowed improving networks and relations among participants. We planned to offer a coffee, although we then considered that as the experiment involved a short period of time, it was not a good idea to move our participants from one place to another. Therefore, we used a classroom with a special area for the coffee break.
3. Our participants valued time as a very positive aspect and basically, as one of the most relevant learned lessons in our experience: to **be on time was essential to collaborate with companies,** because time means money and they usually have very close and complex agendas.

Nevertheless, our experiment not only provided good results; during the execution we addressed two important negative aspects. The first one was related to the number of PCs. We reserved a classroom in the School of Computer Languages and Systems and we had to face up a very simple problem. In the form designed to book classrooms, the School recommended the maximum capacity of each classroom.. Thus, we thought that this figure involved the total number of computers in the classroom as well. In consequence, the place was designed with one computer for two people and we had to divide participants in two different groups. It was important in the session of Wednesday morning since it brought together the largest number of participants. It was really a misfortune that could be avoided, as we performed a test on the previous Monday and we did not realize it. Moreover, we had to change the execution at that moment, and we even used our personal laptops to decrease the effect that issue would cause on the timetable of the experiment. As a result, we were delayed 30 minutes approximately.

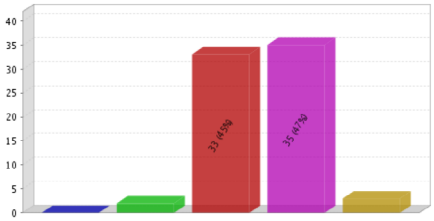
Another drawback was a conceptual decision that we made regarding the definition of the experiment. We considered that it could be interesting to conduct the experiment with people of different level of knowledge: juniors, seniors, managers, and so on. The results obtained with that decision were really good but they entailed a considerable problem in the execution of the experiment, because along the Early Testing course we had a very heterogeneous audience. We reduced the effect of this problem by involving in the project many team members (more than 15 people participated in each session), who assisted and helped some of the youngest participants[10].

[10] It is very important to stick out that help did not mean conducting the experiment, but only solving problems or clarifying concepts related with metamodels, transformations or even

4.3 After Running the Experiment

After conducting the experiment, we also followed some strategies to demonstrate our participants the positive effect that taking part in university experiments may entail.
Obviously, by gathering information, we learned a lot of aspects that we had never thought before about the use of Early Testing as well as how valuable experts considered our tools. However, the analysis of these conclusions is not the aim of this paper. As we introduced, we intend to analyze which strategies are suitable to improve communication with companies in order to get their involvement in software experiments.

Even though we are not going to present the conclusions of our research, Figure 1 presents an interesting graph obtained from OPINA with the answers to a very specific question asked to each participant in the last questionnaire of the experiment: in your view, is this kind of experience developed between university and companies interesting?



Blue: It was not interesting	0%
Green: I consider I did not learn anything	2%
Red: Yes, but I wanted to get more learning	45%
Pink: Yes, I liked it and I learned very much	47%
Yellow: It was wonderful	4%

Fig. 1. Satisfaction percentage

As it can be concluded from this chart, a large number of our participants considered the experiment as very interesting as they thought that collaboration between entities could be very suitable for improving their jobs. They also stated that it was an profitable working day that offered very relevant solutions to enhance their own businesses. For that reason, and taking into account that 98% of participants judged the experience as very relevant for their jobs, we deem it necessary that the research community improves this kind of collaborations.

After the experiment, we also continued applying strategies to keep the interest of our partner companies and participants. Firstly, after the execution of the experiment, we sent **a personal email with the global figures taken out from the experiment** (number of participants, companies, and results, for example), just to demonstrate the significance of the results. **We also sent a letter to Junta de Andalucía and we published a especial news about the success of the experiment in our university social networks**.

Two weeks after conducting the project, each participant received his/her **certificate of participation via email** where they were offered the possibility to be sent the original one. Then, the

functional test definitions. Our team did not influence the experts' execution of the experiment.

team was assessing the results throughout the next three months (December, January and February). Currently, this evaluation has just finished and next week we will send our participants a new email, giving them the possibility to know their results. We guaranteed the confidentiality of the results in the experiment. In consequence, we cannot give information about each participant and his/her performance, but we can reveal their personal results and positions in comparison to other participants. It means that we are going to meet each participant who deems it necessary and, depending on his/her profile, we will compare his/her results with other participant's results with the same profile as well as will give them some advise under our consideration. Besides, they can **freely download examples, tools and manuals** used along the experiments and they can have a direct connection with the research team in case that they want to know more about Early Testing. We have established a direct link to companies as well. This experience has also strengthened the communication with companies we had worked before, by means of demonstrating that the experiment is also relevant for them. For this reason, the results have led to a new phase. We are closing face-to-face meetings with manager teams at companies by offering them a detailed evaluation of the current situation of their experts, and we will freely provide them with a theoretical evaluation of their present situation. Obviously, we have to be very careful with these meetings, as we do not pretend to be conceited. Companies know very well their own businesses, thus, we will only offer a constructive opinion according to our experiment's results.

5 FINAL SUMMARY OF SUCCESSFUL STRATEGIES

To finish and summarize, Table 8 presents learned positive lessons for each phase of the experiment (those failed are presented in the test).

Table 8. Summary schema.

preparation	• To get the support of companies' costumers is essential to attract their interest for the experiment. • To define a very concrete call, with clients sponsoring, as well as a very concrete agenda is critical. The call has to include clearly advantages for companies. • To offer personal invitations (by means of personal emails, phone calls, WhatsApps and some other means of communication). • To hold face-to-face meetings with manager teams and to engage them in the project. • To keep time and schedule arrangements as carefully as possible. Time means money for companies, thus time investments have to be well described.
execution	• To agree with companies on the days and hours for the execution, even if several sessions or "extra hours" need to be scheduled, for instance, on Friday evening or at weekends. • To execute as many tests as possible to check that there is not any technical problem or problems with the infrastructure or facilities. • To prepare personal documentation for each participant. Each participant is important for the experiment, and this is what they should be demonstrated. • To be very strict with time, as it is one of the most critical aspects. • To consider time for coffee as a key point to promote personal relations and future collaborations networks. Companies are also interested in improving and increasing their networks.

analysis	• To send information to participants. The experiment has to offer results and our participants has to know it. • To be sure that you offer all the advantages that you promise in your experiment. • To try to have face to face meeting with companies to learn even more. • To try to explote the network that you can get with an experiment like this us. It is a good way for future works and transference projects.

6 CONCLUSIONS AND FUTURE WORK

This paper has presented the global view of a software experiment executed at the University of Seville in collaboration with the Politecnical University of Madrid and Instituto Tecnológico de Aragón. The paper does not focus on presenting the experiment. In fact, the project has provided interesting feedback to go on researching. However, one of the most relevant aspects of this study is amount of people from companies engaged to collaborate in the experiment. There were 97 experts registered and finally only 76 participated, with a total of 32 companies involved. As the literature demonstrates, to get this number of participants is very difficult. Therefore, this paper tries to present which strategies were used to get this goal. As a future work, we would like to publish and to get information about the experiment and obviously, we aim to improve our tools. Besides, after the success of this first experience, we are looking forward repeating it with other areas of NDT like quality, project management or requirements management. Besides, many companies that took part in the experiment asked to repeat it with all their employees at their own offices or facilities, what makes us be very confident. For instance, companies from Zaragoza, which participated in the prototype, demanded to repeat the experience at their own headquarters.

Finally, this paper will conclude with a global consideration. This study has confirmed that communication and collaboration with companies is possible. Nevertheless, communication has to be in two directions. If we aim to involve companies in our experiments, we will have to consider their situation and availability and we will simplify the process as much as possible, without being detrimental to our scientific method.

ACKNOWLEDGEMENTS

This research has been partially supported by the Megus project (TIN2013-46928-C3-3-R) and by the SoftPLM Network (TIN2015-71938-REDT) of the Spanish Ministry of Economy and Competitiveness.

REFERENCES

[1] ATOS Origin. http://atos.net/en-us/home.html. Last accessed 03/2016.
[2] Consejería de Hacienda y administración Pública. Junta de Andalucía. http://www.juntadeandalucia.es/haciendayadministracionpublica/. Last accessed 03/2016.
[3] Dieste, O., Juristo, N., & Martínez, M. D. 2013. Software industry experiments: A systematic literature review. *In Proceedings of the 1st International Workshop on Conducting Empirical Studies in Industry (pp. 2-8).* IEEE Press.
[4] Enterprise Architect. SparxSystems. Enterprise Architect. Website: www.sparxsystems.com.au. Last accessed 03/2016.

[5] Escalona, M.J. Models and Techniques for the Specification and Analysis of Navigation in Software Systems. 2004. *Ph-thesis. University of Seville*, Spain.
[6] Escalona, M. J., Gutiérrez, J. J., Villadiego, D., León, A., & Torres, J. (2007). Practical experiences in web engineering. In *Advances in Information Systems Development* (pp. 421-433). Springer US.
[7] Escalona, M. J. & Aragón, G. 2008. NDT. A model-driven approach for web requirements. *Software Engineering, IEEE Transactions on*, *34*(3), 377-390.
[8] Escalona, M. J., García-García, J. A., Mas, F., Oliva, M., & Del Valle, C. (2013). Applying Model-Driven Paradigm: CALIPSOneo Experience. In *CAiSE IT* (pp. 25-32).
[9] Fidetia. www.fidetia.es. Last accessed 03/2016.
[10] García-García, J. A., Ortega, M. A., García-Borgoñón, L., & Escalona, M. J. 2012. NDT-Suite: a model-based suite for the application of NDT. In Web Engineering (pp. 469-472).
[11] Gutiérrez, J. J., Escalona, M. J., Mejías, M., Torres, J., & Centeno, A. H. (2008, June). A case study for generating test cases from use cases. In *Research Challenges in Information Science, 2008. RCIS 2008. Second International Conference on* (pp. 209-214). IEEE.
[12] Hotel Ambassador. iwt2.org/actividad-grupo/investigacion/resultados/ndt/aprendiendo-ndt/. Last accessed 03/2016.
[13] IECISA. www.iecisa.com/. Last accessed 03/2016.
[14] Instituto Tecnológico de Aragón. http://www.itainnova.es/. Last accessed 03/2016.
[15] Jedlitschka, A., Juristo, N., & Rombach, D. 2014. Reporting experiments to satisfy professionals' information needs. *In Empirical Software Engineering,* 19(6), 1921-1955.
[16] Jedlitschka, A., & Pfahl, D. 2005, Reporting guidelines for controlled experiments in software engineering. *In International Symposium on Empirical Software Engineering,* (pp. 10-pp). IEEE.
[17] Juristo, N., Moreno, A.M. 2001. Basics of Software Engineering Experimentation .
[18] Martínez-García, A., García-García, J. A., Escalona, M. J., & Parra-Calderón, C. L. (2015). Working with the HL7 metamodel in a Model Driven Engineering context. *Journal of biomedical informatics*, *57*, 415-424.
[19] Opina. opinahq.com/. Last accessed 03/2016.
[20] Object Management Group. Unified Model Language. V2.5. www.omg.org/spec/UML/2.5. Last accessed 03/2016.
[21] Punto de informacion del Flamenco. www.juntadeandalucia.es/cultura/ centroandaluzflamenco/Pif/index.php. Last accessed 03/2016.
[22] Robson, C., & McCartan, K. 2016. Real world research. *Wiley*.
[23] Runeson, P., & Höst, M. 2009. Guidelines for conducting and reporting case study research in software engineering. *In Empirical software engineering*, 14(2), 131-164
[24] Sjøberg, D. I., Hannay, J. E., Hansen, O., Kampenes, V. B., Karahasanovic, A., Liborg, N. K., & Rekdal, A. C. 2005. A survey of controlled experiments in software engineering. *In IEEE Transactions on Software Engineering,* 31(9), 733-753.
[25] Topçu, O., Durak, U., Oğuztüzün, H., & Yilmaz, L. (2016). Model Driven Engineering. In *Distributed Simulation* (pp. 23-38). Springer International Publishing.
[26] Vegas, S., Dieste, Ó., & Juristo, N. 2015. Difficulties in running experiments in the software industry: experiences from the trenches. *In Proceedings of the Third International Workshop on Conducting Empirical Studies in Industry* (pp. 3-9). IEEE Press.
[27] Web Engineering and Early Testing Group. University of Seville 2016. www.iwt2.org. Last accessed 03/2016.
[28] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. 2012. Experimentation in software engineering. *In Springer Science & Business Media*.
[29] Spector, A. Z. 1989. Achieving application requirements. In *Distributed Systems*, S. Mullender, Ed. ACM Press Frontier Series. ACM, New York, NY, 19-33. DOI= http://doi.acm.org/10.1145/90417.90738. Last accessed 03/2016.

Pruebas sobre aplicaciones de bases de datos orientadas a grafos: un enfoque basado en modelos

Raquel Blanco, Javier Tuya

Departamento de Informática, Universidad de Oviedo
Campus de Gijón, s/n, 33204 Gijón-Asturias
{rblanco, tuya}@uniovi.es

Resumen. Las bases de datos NoSQL plantean nuevos desafíos a la hora de probar las aplicaciones que las utilizan, debido a que sus modelos de datos y sus modos de acceso difieren de las bases de datos relacionales. La prueba de aplicaciones que acceden a bases de datos relacionales ha atraído el interés de muchos investigadores, mientras que la prueba de aplicaciones que acceden a bases de datos NoSQL es un área que aún no ha sido prácticamente explorada. Este trabajo describe un enfoque que permite crear modelos que definen objetivos de prueba para aplicaciones que utilizan bases de datos orientadas a grafo, a partir de la especificación del sistema y de un modelo de datos conceptual. Estos modelos son empleados posteriormente para derivar los requisitos de prueba que guiarán la generación de los casos de prueba. El enfoque ha sido aplicado a una aplicación que representa un problema del mundo real y los resultados muestran que permite diseñar casos de prueba capaces de detectar fallos que pueden aparecer tanto en la especificación del sistema como en la implementación.

Palabras clave. Pruebas sobre bases de datos orientadas a grafos, pruebas basadas en la especificación, base de datos de prueba, model-based testing.

1 Introducción

Las bases de datos son probablemente el recurso más importante de una organización y constituyen la parte central de muchos sistemas software. En la actualidad, muchas organizaciones necesitan recopilar y almacenar cantidades cada vez más vastas de información y requieren un acceso a la misma altamente eficiente. Esto ha propiciado una nueva era en el desarrollo de tecnologías y modelos de datos para almacenar información, englobado todo ello dentro del término *bases de datos NoSQL* (*Not Only SQL*), las cuales están ganando mercado de forma creciente [16].

Existe un gran número de tecnologías NoSQL (actualmente más de 225) [22], que se clasifican en cuatro categorías, de acuerdo a su modelo de datos [20]: clave-valor, basadas en documentos, familia de columnas y orientadas a grafos. A pesar de sus diferencias, estos tipos de bases de datos tienen algo en común: no requieren un esquema explícito, lo que supone una marcada diferencia con las bases de datos relacionales.

Probar las aplicaciones que acceden a bases de datos NoSQL supone nuevos desafíos por diversos motivos. Por un lado, la falta de estandarización de los lenguajes de consulta hace que cada tecnología proporcione sus propias APIs y lenguajes de consulta, los cuales no son tan ampliamente conocidos como SQL. Además, la programación de consultas complejas puede resultar difícil [16], en particular las consultas en las bases de datos orientadas a grafos pueden resultar difíciles de escribir, entender y mantener [1]. Asimismo, al no mantenerse las propiedades ACID, sino las propiedades BASE [24], pueden ocurrir pérdidas de fiabilidad y/o consistencia.

Por otro lado, a pesar de no requerir un esquema explícito en la base de datos, las aplicaciones normalmente cuentan con un modelo conceptual subyacente que representa los datos que manejan (en adelante, *modelo de datos conceptual*). La falta de un esquema en la base de datos implica que no se establecen restricciones para el almacenamiento físico de los datos, lo que puede ocasionar que estos no satisfagan el modelo de datos conceptual de la aplicación y puedan provocar funcionamientos erróneos en la aplicación y/o salidas incorrectas hacia el usuario.

La problemática asociada a las pruebas sobre aplicaciones que acceden a bases de datos ha sido abordada en múltiples trabajos, bajo diversas líneas de investigación. Así, se han desarrollado criterios de suficiencia bajo diferentes enfoques: basados en el código procedural y las sentencias SQL que contienen [7], específicamente creados para manejar el código SQL [11, 12, 26, 28, 31] o basados en la especificación [3], entre otros. Asimismo, se han desarrollado herramientas para automatizar los criterios de suficiencia, como por ejemplo [13, 25, 34]. La generación de datos de prueba también ha sido abordada en varios trabajos, abarcando la generación de la base de datos de prueba [2, 17, 32] y la generación tanto de la base de datos de prueba como de las entradas del programa [5, 18], mientras que la reducción de la base de datos de prueba de forma que contenga información significativa ha sido afrontada en [27, 29]. Otros trabajos han centrado su investigación en la prueba del esquema de la base de datos, desarrollando para ello criterios de suficiencia [19, 33], técnicas para la generación de datos de prueba [14] y métodos para la priorización de casos de prueba cuando el esquema de la base de datos cambia [9, 10].

Sin embargo, estos trabajos dependen del uso de sentencias SQL y/o de la existencia de un esquema implícito en la base de datos, por lo que no pueden ser aplicados directamente a la prueba de aplicaciones que acceden a bases de datos NoSQL. Por ello, es necesario desarrollar nuevos enfoques o realizar adaptaciones de los existentes para probar este tipo de aplicaciones, donde se tengan en cuenta los nuevos modelos de datos y las características específicas de cada tecnología NoSQL.

El objetivo de este trabajo es desarrollar un enfoque automatizable que permita probar aplicaciones que acceden a bases de datos orientadas a grafos, considerando las características del modelo de datos de este tipo de tecnología: los datos se almacenan en nodos y en las relaciones entre ellos, y tanto nodos como relaciones pueden contener propiedades. Las bases de datos orientadas a grafos están ganando popularidad y miles de organizaciones las utilizan en aplicaciones de logística, detección de fraude, gestión de acceso e identidad, etc. [21].

Para lograr este objetivo, nuestro trabajo utiliza un enfoque *model-based testing*, el cual ha sido empleado por diversos autores para probar aplicaciones que acceden a bases de datos relacionales (por ejemplo, [3, 6, 8, 15]). En *model-based testing*, el sistema es modelado para identificar los aspectos importantes a probar, obteniendo así un *modelo de pruebas*. A partir de dicho modelo de pruebas se pueden derivar automáticamente los requisitos de prueba, aplicando un criterio de suficiencia, y posteriormente se pueden obtener los casos de prueba utilizando una determinada técnica de generación [30]. En nuestro trabajo previo [4] se propuso la integración de *model-based testing* dentro del paradigma MDA (*model-driven architecture*) para hacer frente a la automatización del proceso de prueba sobre aplicaciones que acceden a bases de datos orientadas a grafos, y se describió una versión inicial del modelo de pruebas, el cual se diseña a partir de la especificación del sistema y del modelo de datos conceptual de la aplicación. El presente trabajo extiende y describe con mayor profundidad el modelo de pruebas de nuestro trabajo previo.

El resto del trabajo se ha organizado en las siguientes secciones: la sección 2 describe el modelo de pruebas para aplicaciones que acceden a bases de datos orientadas a grafos y la sección 3 lo caracteriza mediante una taxonomía; la sección 4 presenta los resultados del caso de estudio. El artículo finaliza con las conclusiones y el trabajo futuro.

2 Modelo de pruebas para aplicaciones que acceden a bases de datos orientadas a grafos

El enfoque presentado en este trabajo permite crear modelos que definen objetivos de prueba a partir de la especificación de la aplicación y del modelo de datos conceptual de la misma. Estos modelos, denominados *modelos de pruebas*, se componen de uno o varios escenarios, denominados *test views*, que indican los nodos y relaciones a considerar en la base de datos para probar aspectos importantes del comportamiento de la aplicación.

La creación de esta base de datos, denominada *base de datos de prueba*, de forma que contenga datos significativos es un factor crucial en el diseño de los casos de prueba, puesto que estos datos, además de transformarse para producir la salida de la prueba, tienen que representar las situaciones de interés a probar, para que así el programa bajo prueba las pueda ejercitar. Este trabajo se centra en la definición de *test views* para pruebas unitarias, las cuales están especialmente orientadas a la posterior creación de la base de datos de prueba.

En la Figura 1 se muestra el metamodelo de nuestro enfoque, donde aparecen representados los elementos que componen una *test view* (representada por la metaclase *TestView*). Estos elementos son:

- *View node* o *vNode* (representado por la metaclase *ViewNode*): tipo de nodo de la base de datos orientada a grafos, el cual se deriva a partir de una entidad del modelo de datos conceptual. Cada *vNode* se etiqueta para indicar qué entidad está representando (atributo *label*).

- *View path* o *vPath* (representado por la metaclase *ViewPath*): camino dirigido que relaciona dos *vNodes*, de acuerdo a una semántica derivada de las relaciones del modelo conceptual de datos (atributo *semantic*). Los *vPaths* pueden ser de dos tipos: permitido o no permitido (representados por la metaclases *AllowedViewPath* y *NotAllowedViewPath*, respectivamente). Un *vPath* permitido indica que el camino puede aparecer en la base de datos, mientras que un *vPath* no permitido expresa que el camino no debería aparecer en la misma.
- *Mock path* (representado por la metaclase *MockPath*): camino que no está completamente definido. El objetivo de la prueba no está centrado en un camino específico entre dos *vNodes*, sino en que éste exista. La longitud de dicho camino puede ser importante para probar el correcto comportamiento de la aplicación, por ello, un *mock path* puede definirse con una longitud mínima y máxima (atributos *minLength* y *maxLength*, respectivamente).
- *vPath constraint* (representada por la metaclase *ViewPathConstraint*): restricción sobre un grupo de *vPaths* que condiciona si cada uno de ellos puede, no puede o debe aparecer al mismo tiempo que los demás en la base de datos. Las *vPath constraints* pueden ser de tres tipos: *XOR*, que indica que sólo uno de los *vPaths* permitidos puede aparecen en la base de datos; *OR*, que indica que varios *vPaths* permitidos pueden aparecer de forma simultánea en la base de datos; *AND*, que indica que todos los *vPaths* del grupo deben aparecer al mismo tiempo en la base de datos. Los *vPaths* no permitidos no deberían aparecer en la base de datos.
- *vPath connector* o *connector* (representado por la metaclase *ViewPathConnector*): agrupa un conjunto de *vPaths* que están restringidos por la misma *vPath constraint*. Un *connector* puede agrupar *vPaths* que comienzan o terminan en el mismo *vNode*.
- *View property* o *vProperty* (representada por la metaclase *ViewProperty*): propiedad de un *vNode* o de un *vPath* que se deriva de los atributos de las entidades y relaciones del modelo de datos conceptual.
- *View business rule* o *vRule* (representada por la metaclase *ViewBusinessRule*): regla de negocio que impone condiciones sobre una o varias *vProperties*, de acuerdo a la especificación del sistema, para formar uno o varios predicados (atributo *predicates*). Una *vRule* también define el dominio de datos sobre el cual va a ser aplicada, es decir, el conjunto de nodos y relaciones de la base de datos afectados por los predicados. A este dominio de datos se le denomina *contexto* y está representado por el atributo *context*. La definición de las *vRules*, basada en el trabajo [3] se encuentra fuera del alcance del presente artículo.

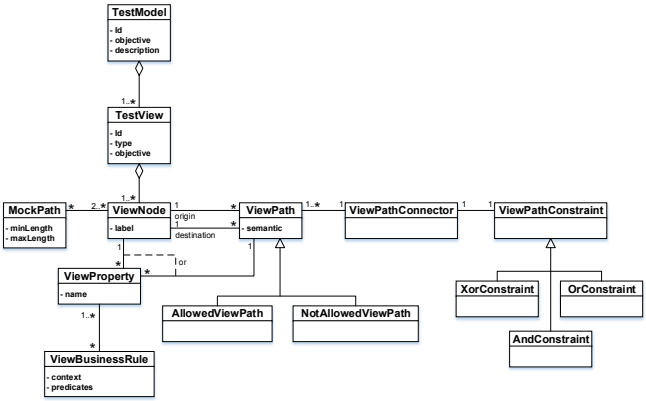


Figura 1. Metamodelo de pruebas para aplicaciones con bases de datos orientadas a grafos

Para ilustrar la creación de *test views* y su notación gráfica, vamos a utilizar como ejemplo una aplicación que determina el nivel de riesgo de padecer una enfermedad de acuerdo a diferentes factores tales como la gravedad de los episodios previos sufridos por el paciente (clasificados en 3 niveles), la existencia de la enfermedad en su familia, etc. El modelo de datos conceptual se muestra en la Figura 2.

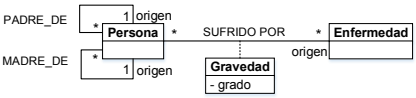
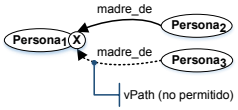


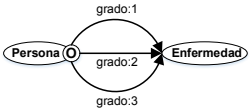
Figura 2. Modelo de datos conceptual de la aplicación que determina el riesgo de padecer una enfermedad

Basándose en la especificación de dicha aplicación y en el modelo de datos conceptual, algunos escenarios interesantes serían: (a) probar la multiplicidad de la relación "madre_de", (b) probar la aparición de varios episodios de una enfermedad sufridos por la misma persona con diferentes niveles de gravedad, (c) probar la aparición de varios episodios de una enfermedad en personas de la misma familia. Las *test views* correspondientes a estos escenarios, junto con la identificación de los elementos que las componen, se muestran en la Figura 3. En esta figura se puede ver que cuando un tipo de nodo tiene varias instancias en una *test view*, éste genera varios *vNodes* identificados por la etiqueta $entidad_i$, donde *entidad* se corresponde con la entidad del modelo de datos conceptual representada por el *vNode* y el subíndice hace referencia al número de instancia del mismo. Por ejemplo, los *vNodes* "$Persona_1$", "$Persona_2$" y "$Persona_3$" son

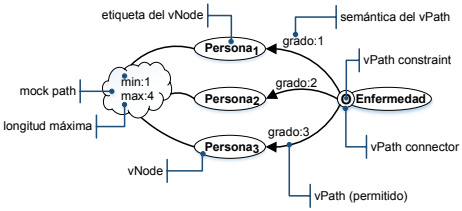
tres instancias diferentes del mismo tipo de nodo "Persona". Respecto a las *vPath constraints*, el tipo *XOR* se representa con "X", el tipo *OR* se representa con "O" y el tipo *AND* se representa con "+".



(a) multiplicidad de la relación "madre_de"



(b) episodios en la misma persona



(c) episodios en la misma familia

Figura 3. Ejemplos de *test views*

La *test view* de la Figura 3(a) indica que el *vPath* entre "Persona$_2$" y "Persona$_1$" puede aparecer en la base de datos, mientras que el *vPath* entre "Persona$_3$" y "Persona$_1$" no puede estar presente. La Figura 3(b) indica que varios *vPaths*, los cuales representan diferentes niveles de gravedad de una enfermedad, pueden aparecer de forma simultánea entre una instancia de "Persona" y una instancia de "Enfermedad". En cuanto a la Figura 3(c), ésta indica que en la base de datos pueden aparecer varios *vPaths* representando episodios de distinta gravedad de una enfermedad en tres personas diferentes (instancias "Persona$_1$", "Persona$_2$" y "Persona$_3$"), las cuales tienen una relación familiar entre grado 1 y grado 4. Esta relación familiar, no totalmente definida, está representada por el *mock path*.

Una vez se han definido las *test views* que componen el modelo de pruebas, se pueden derivar los requisitos de prueba aplicando un determinado criterio de suficiencia sobre dichas *test views*. Estos requisitos de prueba permiten guiar la generación de los casos de prueba, donde la entrada estará formada por el estado de la base de datos antes de la ejecución del caso de prueba (base de datos de prueba) y los valores suministrados por el usuario, y la salida esperada estará formada por el estado de la base de datos después de la ejecución del caso de prueba y los valores mostrados al usuario.

3 Taxonomía

Las *test views* del enfoque presentado en este trabajo están caracterizadas por una taxonomía de tres dimensiones, la cual se muestra en la Figura 4.

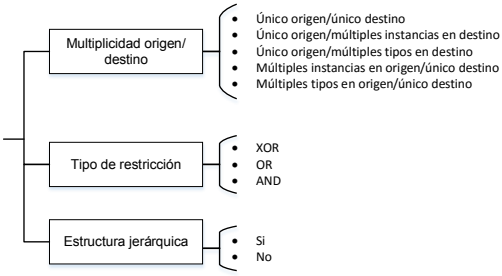


Figura 4. Taxonomía de las *test views*

La dimensión *multiplicidad origen/destino* indica el número de *vNodes* origen y destino relacionados por un conjunto de *vPaths* agrupados por un *connector*. Se clasifica en cinco alternativas:

- *Único origen/único destino*: todos los *vPaths* tienen el mismo *vNode* origen y el mismo *vNode* destino.
- *Único origen/múltiples instancias en destino*: todos los *vPaths* tienen el mismo *vNode* origen, pero cada uno de ellos finaliza en un *vNode* diferente. Todos los *vNodes* destino son instancias del mismo tipo de nodo.
- *Único origen/múltiples tipos en destino*: todos los *vPaths* tienen el mismo *vNode* origen, pero cada uno de ellos finaliza en un *vNode* diferente. Cada *vNode* destino representa un tipo diferente de nodo.
- *Múltiples instancias en origen/único destino*: todos los *vPaths* tienen el mismo *vNode* destino, pero cada uno de ellos comienza en un *vNode* diferente. Todos los *vNodes* origen son instancias del mismo tipo de nodo.
- *Múltiples tipos en origen/único destino*: todos los *vPaths* tienen el mismo *vNode* destino, pero cada uno de ellos comienza en un *vNode* diferente. Cada *vNode* origen representa un tipo diferente de nodo.

La dimensión *tipo de restricción* hace referencia al tipo de *vPath constraint* que restringe a un conjunto de *vPaths* (permitidos y no permitidos), los cuales están agrupados mediante un *connector*. Se divide en tres categorías, que se corresponden con los tres tipos de *vPath constraints* descritos previamente: *XOR*, *OR*, *AND*.

La dimensión *estructura jerárquica* establece si entre un grupo de *vNodes* origen (o destino) existe un *mock path* que los relaciona. Esta dimensión se puede combinar con todas las categorías de la dimensión *multiplicidad origen/destino*, excepto con *único origen/único destino*.

Basándose en esta taxonomía, se pueden definir diferentes tipos de *test views*, las cuales se denominan *basic test views* dado que pueden ser combinar para diseñar objetivos de prueba más complejos. Una *test view* que está compuesta por varias *basic test views* se denomina *complex test view*.

Por ejemplo, la *test view* de la Figura 3(c) es una *basic text view* caracterizada por la multiplicidad *único origen/múltiples instancias en destino*, el tipo de restricción *OR* y la existencia de una estructura jerárquica.

4 Caso de estudio

El enfoque presentado en este trabajo ha sido aplicado a un ejemplo de una aplicación del mundo real denominada "*autorización y control de acceso*" [23]. Esta aplicación representa el negocio de una compañía de servicios de comunicaciones, que ofrece a las organizaciones que la contratan la posibilidad de auto-gestionar sus cuentas. Los administradores de dichas organizaciones pueden añadir y eliminar servicios en nombre de sus empleados. Para asegurar que los recursos sólo son vistos y modificados por los usuarios autorizados se ha diseñado un complejo sistema de control de acceso, que gestiona diferentes tipos de permisos y estructuras jerárquicas entre organizaciones. El modelo de datos conceptual se muestra en la Figura 5.

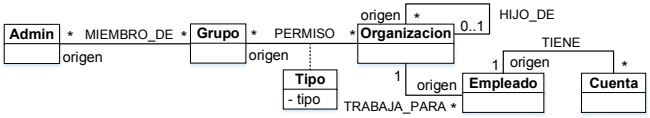


Figura 5. Modelo de datos conceptual de la aplicación "*autorización y control de acceso*"

Los administradores pueden ser miembros de varios grupos, los cuales pueden tener diversos permisos sobre una estructura de organizaciones. Cada organización puede tener varias organizaciones hijas, con sus propios empleados y cuentas. Se han definido tres tipos de permisos entre grupos y organizaciones: (1) *permitido_herencia* permite a los administradores del grupo gestionar las cuentas de la organización y las de las organizaciones que descienden de ella a lo largo de la estructura jerárquica definida; (2) *permitido_sin_herencia* permite a los administradores del grupo gestionar las cuentas de la organización, pero no las de sus descendientes en la estructura jerárquica; (3) *denegado* prohíbe a los administradores del grupo la gestión de las cuentas de la organización y las de sus descendientes. El sistema de control de acceso también establece una serie de precedencias entre permisos, puesto que un administrador puede ser miembro de varios grupos con diferentes permisos sobre la misma organización. Así, el permiso *denegado* prevalece sobre el permiso *permitido_herencia* y el permiso *permitido_sin_herencia* prevalece sobre el permiso *denegado*.

La especificación de esta aplicación define tres consultas sobre la base de datos orientada a grafos para hallar todas las cuentas accesibles para un administrador, para

determinar si un administrador tiene acceso a una determinada cuenta y para hallar todos los administradores que pueden acceder a una cuenta.

En primer lugar, se diseñaron una serie de *test views* de acuerdo a la especificación del sistema. La Figura 6 muestra una de las *test views*, cuyo objetivo es probar el efecto de la herencia de los distintos tipos de permisos. Para ello se representa que un grupo puede tener permisos distintos sobre diferentes organizaciones, las cuales están organizadas jerárquicamente. Además, se indican las longitudes mínima y máxima de la estructura jerárquica a considerar en las pruebas. La semántica "sin_especificar" del *vPath* que relaciona "Grupo" con "Organización$_4$" representa que el grupo no tiene un permiso explícito sobre dicha organización.

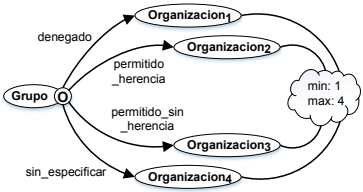


Figura 6. *Test view* de la aplicación "autorización y control de acceso"

A continuación, se derivaron los requisitos de prueba de forma automática a partir de las *test views*, mediante un script que implementa un criterio de suficiencia basado en una técnica combinatoria y valores límite. Dicho criterio realiza permutaciones sin repetición sobre los *vNodes* relacionados con el *mock path*, de forma que se generan distintos órdenes jerárquicos entre ellos, y aplica la técnica de valores límite sobre las longitudes mínima y máxima del *mock path* para ejercitar diferentes niveles de profundidad de la herencia entre cada par de *vNodes*. La Figura 7 muestra tres de los requisitos de prueba obtenidos. Ahora los *mock path* son caminos dirigidos para indicar la estructura jerárquica concreta que se está representado y están etiquetados con la longitud que debe tener dicho camino cuando se generen los casos de prueba.

Posteriormente, a partir de los requisitos de prueba se generó la base de datos de prueba, considerando una base de datos orientada a grafos concreta (en nuestro caso Neo4j [21]). La Figura 8 muestra los nodos y relaciones que se introdujeron en la base de datos de prueba para cubrir los requisitos de prueba de la Figura 7. Los nodos "G1", "O1 a "O4", junto con sus respectivas relaciones, cubren el requisito de prueba de la Figura 7(a). Los nodos "G1", "O5" a "O8" y las relaciones que los unen cubren el requisito de prueba de la Figura 7(b). Los nodos "G1", "O9" a "O20" y las relaciones entre ellos cubren una parte de los requisitos de prueba de la Figura 7(c) y de la Figura 7(d), concretamente las longitudes de los caminos que deben existir entre "Organizacion$_2$" y "Organizacion$_3$" de la Figura 7(c) (nodos "O9" a "O14") y entre "Organizacion$_1$" y "Organización$_2$" de la Figura 7(d) (nodos "O15" a "O20"). El resto de nodos y relaciones se introdujeron en la base de datos de prueba para satisfacer el modelo de datos conceptual. Finalmente, se generó el código de prueba a ser ejecutado, utilizando

los lenguajes Cypher y Java. Actualmente, tanto la base de datos de prueba como el código de prueba se generan manualmente, sin embargo, estas tareas serán automatizadas en el futuro.

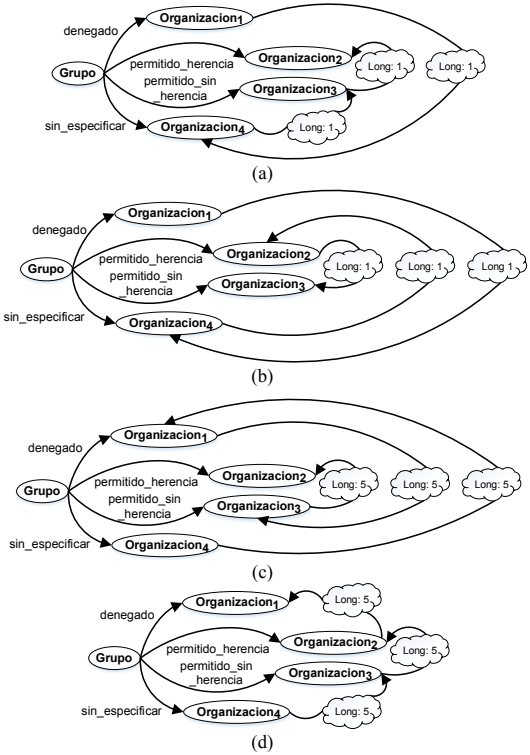


Figura 7. Requisitos de prueba de la aplicación "autorización y control de acceso"

La ejecución de los casos de prueba que utilizan como entrada la base de datos de prueba generada indican que el administrador "A1" puede acceder a las cuentas "AC1", AC2", AC3", "AC5", "AC6", "AC7", "AC9", "AC10", "AC11", "AC12", "AC14" y "AC20", pero ¿realmente debería tener acceso a las cuentas "AC3", "AC6", "AC7" y "AC20"?, ¿debería tener acceso también a la cuenta "AC13"?

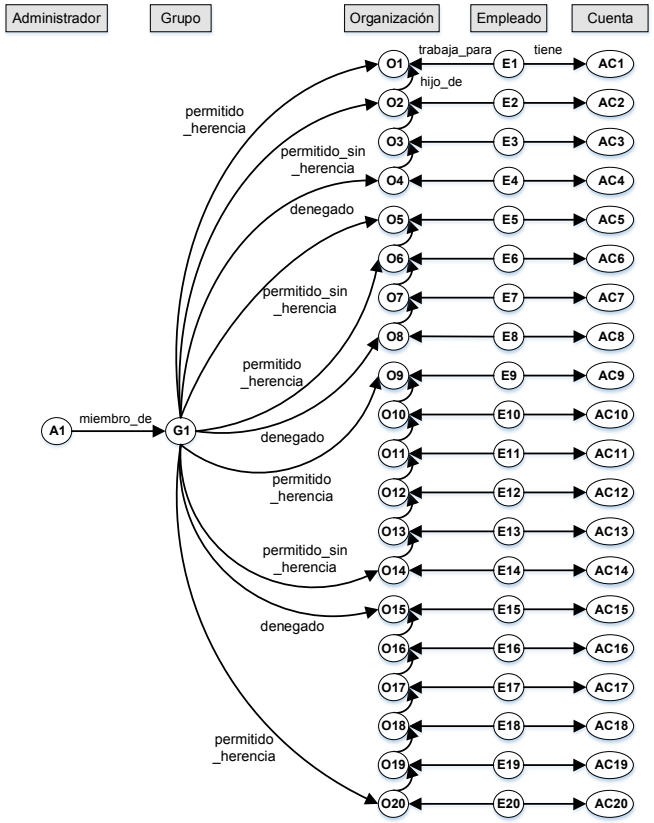


Figura 8. Extracto de la base de datos de prueba de la aplicación "autorización y control de acceso"

Consideremos en primer lugar la cuenta "AC3", que pertenece a la organización "O3". Si se analiza la estructura jerárquica para esta organización se puede ver que desciende de la organización "O2", sobre la cual el grupo "G1" tiene el permiso "permitido_sin_herencia" que otorga acceso a las cuentas de "O2" pero no a las de sus descendientes. En base a esta especificación el administrador "A1" no puede acceder a

"AC3", por lo que se ha encontrado un fallo. Sin embargo, la organización "O2" es hija de la organización "O1" y el grupo "G1" tiene establecido sobre ella el permiso "permitido_herencia", que da acceso a las cuentas de "O1" y a las de todos sus descendientes (entre los que se encuentra "O3"). Por tanto, ¿realmente se ha encontrado un fallo?, ¿qué permiso debe prevalecer? No lo podemos saber, puesto que la especificación no indica la preferencia entre los permisos "permitido_herencia" y "permitido_sin_herencia". Esa misma incertidumbre está presente también para las cuentas "AC6" y "AC7". Por tanto, la especificación tiene un defecto, pues es incompleta. Si la salida observada es igual a la esperada, sólo la especificación tiene un defecto, mientras que si la salida observada no es igual a la esperada, tanto la especificación como la implementación tienen un defecto.

En cuanto a la cuenta "AC13", ésta pertenece a la organización "O13", la cual es descendiente de la organización "O9" sobre la que el grupo "G1" tiene el permiso "permitido_herencia". Puesto que a lo largo de la estructura jerárquica entre "O9" y "O13" no se establece ningún permiso que evite el acceso a las cuentas de las organizaciones, el administrador "A1" debería tener acceso a "AC13". Por otro lado, la cuenta "AC20" pertenece a la organización "O20", que es descendiente de la organización "O15" sobre la que "G1" tiene el permiso "denegado". Dado que "G1" no tiene el permiso "permitido_sin_herencia" sobre "O20" para anular la denegación de acceso a sus cuentas, "A1" no puede acceder a "AC20". Por tanto, se han encontrado dos fallos debido a la implementación. Si se analiza el código de la aplicación, se puede ver que las consultas limitan la profundidad de la jerarquía entre organizaciones, sin embargo la especificación no se establece ningún límite al respecto.

5 Conclusiones y trabajo futuro

Este trabajo presenta un enfoque basado en *model-based testing* para probar la funcionalidad de las aplicaciones que acceden a bases de datos orientadas a grafos. Para ello se define un modelo de pruebas, considerando la especificación del sistema y el modelo de datos conceptual de la aplicación, el cual está compuesto por varias *test views* que representan escenarios interesantes a probar.

Los resultados del caso de estudio muestran que los casos de prueba obtenidos permiten detectar fallos provocados por defectos debidos a omisiones o errores cometidos en la especificación del sistema o en la implementación. Cabe destacar que una especificación incompleta puede provocar defectos en las aplicaciones, ya que los desarrolladores pueden interpretar de forma errónea lo que el sistema debe hacer.

El trabajo futuro incluye varias líneas. Por un lado, la definición de criterios de suficiencia que consideren las características de las *test views* para derivan automáticamente los requisitos de prueba y el desarrollo de técnicas para generar la base de datos de prueba de forma automática. Actualmente, los requisitos de prueba pueden ser obtenidos automáticamente, pero las *test views* contienen características que aún no se han considerado para realizar dicha generación. Por otro lado, la definición de estrategias de prueba que permitan guiar el diseño de las *test views* y automatizar la generación de algunas de ellas.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia y Tecnología de España (TIN2013-46928-C3-1-R), el Principado de Asturias (GRUPIN14-007) y los fondos FEDER.

Referencias

1. Barmpis, K., Kolovos, D.S.: Evaluation of Contemporary Graph Databases for Efficient Persistence of Large-Scale Models. Journal of Object Technology, 13(2), pp 3:1-26 (2014).
2. Binnig, C., Kossmann, D., Lo, E.: MultiRQP - Generating test databases for the functional testing of OLTP applications. In Proc. of the 1st Int'l Workshop on Testing Database Systems (2008).
3. Blanco, R., Tuya, J., Seco, R.V.: Test adequacy evaluation for the user-database interaction: a specification-based approach. In Proc. of the 5th Int'l Conference on Software Testing, Verification and Validation, pp. 71-80 (2012).
4. Blanco, R., Tuya, J.: A test model for graph database applications: an MDA-based approach. In Proc. of 6th Int'l Workshop on Automating Test Case Design, Selection and Evaluation, pp. 8-15 (2015).
5. Chays, D., Deng, Y., Frankl, P.G., Dan, S., Vokolos, F.I., Weyuker, E.J.: An AGENDA for testing relational database applications. Software Testing, Verification and Reliability, 14(1), 17-44 (2004).
6. de la Riva, C., Suárez-Cabal, M.J., Tuya, J.: Constraint-based test database generation for SQL queries. In Proc. of the 5th Int'l Workshop on Automation of Software Test, pp. 67-74 (2010).
7. Emmi, M., Majumdar, R., Sen, K.: Dynamic Test input generation of database applications. In Proc. of the Int'l Symposium on Software Testing and Analysis, pp. 151-162 (2007).
8. Fujiwara, S., Munakata, K., Maeda, Y., Katayama, A., Uehara, T.: Test data generation for web application using a UML class diagram with OCL constraints. Innovations in Systems and Software Engineering, 7(4), 275-282 (2011).
9. Gardikiotis, S.K., Malevris, N.: A Two-folded Impact Analysis of Schema Changes on Database Applications. International Journal of Automation and Computing, 6(2) 109-123 (2009).
10. Garg, D., Datta A.: Test Case Priorization due to Database Changes in Web Applications. In Proc. of the 5th Int'l Conference on Software Testing, Verification and Validation, pp. 726-730 (2012).
11. Halfond, W.G.J., Orso, A.: Command-form coverage for testing database applications. In Proc. of the 21st IEEE/ACM Int'l Conference on Automated Software Engineering, pp. 69-80 (2006).
12. Kapfhammer, G.M., Soffa, M.L.: A family of test adequacy criteria for database-driven applications. In Proc. of the 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT Int'l Symposium on the Foundations of Software Engineering, pp. 98-107 (2003).
13. Kapfhammer, G.M., Soffa M.L.: Database-aware test coverage monitoring. In Proc. of the 1st India Software Engineering Conference, pp. 77-86 (2008).

14. Kapfhammer, G.M., McMinn, P., Wright, C.J.: Search-Based Testing of Relational Schema Integrity Constraints Across Multiple Database Management Systems. In Proc. of the 6th Int'l Conference on Software Testing, Verification and Validation, pp. 31-40 (2013).
15. Khalek, S.A., Elkarablieh, B., Laleye, Y.O., Khurshid, A.: Query-aware test Generation using a relational constraint solver. In Proc. of the 23rd IEEE/ACM Int'l Conference on Automated Software Engineering, pp. 238-247 (2008).
16. Leavitt, N.: Will NoSQL databases live up to their promise? IEEE Computer, 43(2) 12-14 (2010).
17. Lo, E., Binnig, C., Kossmann, D., Özsu, M.T., Hon, W.K.: A framework for testing DBMS features. The VLDB Journal, 19(2), pp. 203-230 (2010).
18. Marcozzi, M., Vanhoof, W., Hainaut, J.L.: A relational symbolic execution algorithm for constraint-based testing of database programs. In Proc. of the 13th Int'l Working Conference on Source Code Analysis and Manipulation, pp. 179-188 (2013).
19. McMinn, P., Wright, C.J., Kapfhammer, G.M.: The effectiveness of test coverage criteria for relational database schema integrity constraints. ACM Transactions on Software Engineering and Methodology, 25(1) 8:1-8:49 (2015).
20. Moniruzzaman, A.B.M., Hossain, S.H.: NoSQL Database: New Era of Databases for Big data Analytics-Classification, Characteristics and Comparison. International Journal of Database Theory and Application, 6(4) 1-14 (2013).
21. Neo4J, http://neo4j.com
22. NoSQL databases, http://nosql-database.org
23. Robinson, I., Webber, J., Eifrem, E.: Graph databases. O'Reilly (2013).
24. Roe, C.: ACID vs. BASE: The Shifting pH of Database Transaction Processing. www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/ (2012).
25. Tuya, J., Suárez-Cabal M.J., de la Riva, C.: SQLMutation: a tool to generate mutants of SQL database queries. In Proc. of the Second Workshop on Mutation Analysis (2006).
26. Tuya, T., Suárez-Cabal, M.J., de la Riva, C.: Mutating database queries. Information and Software Technology, 49(4) 398-417 (2007).
27. Tuya, J, Suarez-Cabal, M.J., de la Riva, C.: Query-Aware Shrinking Test Databases. In Proc. Second Int'l Workshop on Testing Database Systems (DBTest'09) (2009).
28. Tuya, T., Suárez-Cabal, M.J., de la Riva, C.: Full predicate coverage for testing SQL database queries. Software Testing Verification and Reliability, 20(3) 237-288 (2010).
29. Tuya, J., de la Riva, C., Suárez-Cabal, M.J., Blanco, R.: Coverage-aware test database reduction. IEEE Transactions on Software Engineering, doi: 10.1109/TSE.2016.2519032 (en prensa) (2016).
30. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approach. Software Testing, Verification and Reliability, 22(5) 297-312 (2012).
31. Willmor, D., Embury, S.M. Exploring test adequacy for database systems. In Proc. of the 3rd UK Software Testing Research Group, pp. 123-133 (2005).
32. Willmor, D., Embury, S.M.: Testing the implementation of business rules using intensional database tests. In Proc. of Testing: Academic & Industrial Conference on Practice and Research Techniques, pp. 115-126 (2006).
33. Wright, C.J., Kapfhammer, G.M., McMinn, P.: Efficient Mutation Analysis Of Relational Database Structure Using Mutant Schemata And Parallelisation. In Proc. of the 6th Int'l Conference on Software Testing, Verification and Validation Workshops, pp. 63-72 (2013).
34. Zhou, C., Frankl, P.: JDAMA: Java Database Application Mutation Analyzer. Software Testing, Verification and Reliability, 21(3), 241-263 (2011).

Generación y Ejecución de Escenarios de Prueba para Aplicaciones MapReduce

Jesús Morán, Claudio de la Riva, Javier Tuya

Departamento de Informática, Universidad de Oviedo, Gijón, España
moranjesus@lsi.uniovi.es, {claudio, tuya}@uniovi.es

Resumen. Los programas que procesan grandes cantidades de datos se suelen ejecutar sobre una infraestructura distribuida, como es el caso de las aplicaciones implementadas bajo el modelo de procesamiento MapReduce. Estos modelos permiten al desarrollador centrarse en la funcionalidad de la aplicación y abstraer aspectos relacionados con la infraestructura en la que se ejecutará. Sin embargo, su configuración y estado pueden causar que ciertos defectos sean difíciles de detectar debido a que las pruebas se suelen ejecutar en un entorno controlado, con bajo nivel de paralelismo y con pocos datos de entrada. En este artículo se elabora una técnica de prueba que partiendo de unos datos de entrada, genera y reproduce diferentes configuraciones de la infraestructura con el objetivo de detectar aquellas que revelen defectos funcionales en la aplicación. Esta técnica se automatiza en un motor de ejecución de pruebas y se aplica a un caso de estudio que actualmente se encuentra en producción. Como resultado, partiendo de un caso de prueba de tamaño reducido, se han identificado automáticamente varias configuraciones de la infraestructura que ocasionarían fallos de la aplicación.

Palabras clave: Pruebas del software, MapReduce, Big Data Engineering

1 Introducción

Ante las nuevas necesidades de procesamiento masivo de datos en paralelo han surgido un conjunto de tecnologías de datos y modelos de procesamiento que conforman lo que se denomina *Big Data Engineering* [1]. Entre estos destaca *MapReduce* [2] que permite analizar grandes cantidades de datos basándose en el principio de "divide y vencerás". Estos programas se ejecutan en dos fases sobre una infraestructura distribuida: la fase Mapper divide el problema en varios subproblemas, y a continuación la fase Reducer resuelve cada uno de ellos. Es habitual que estos programas se ejecuten en varios computadores con diferentes recursos y características. Para facilitar la gestión de esta compleja infraestructura se emplean frameworks, destacando *Hadoop* [3] por su implantación en las organizaciones [4].

Desde el punto de vista del desarrollador, los programas *MapReduce* se pueden implementar de forma independiente a la infraestructura mediante el desarrollo de las funcionalidades Mapper y Reducer. Para ello, el framework que gestiona la infraestructura se encarga automáticamente de ejecutar el programa sobre varios computado-

res y controlar el procesamiento de los datos desde la entrada hasta la salida. Entre otros, se divide la entrada en varios subconjuntos de datos, se procesan en paralelo y se re-ejecutan partes del programa cuando sea necesario.

A pesar de que el desarrollador puede crear el programa abstrayéndose de la infraestructura, tiene que considerar cómo ésta puede afectarle a la funcionalidad. En un trabajo anterior [5] se han detectado y clasificado varios defectos funcionales que dependen de cómo la configuración de la infraestructura afecta a la ejecución del programa e influye en su resultado. Estos defectos suelen enmascararse durante la ejecución de las pruebas ya que éstas se ejecutan sobre una configuración que no tiene en cuenta las diferentes situaciones que pueden producirse en un entorno de producción, como por ejemplo el nivel de paralelismo o posibles los fallos en la infraestructura [6]. Por otra parte, aunque las pruebas se ejecuten en el entorno similar al de producción, algunos defectos podrían no ser detectables, ya que es habitual que las entradas de prueba tengan pocos datos, lo que conlleva que no sea necesario paralelizar la ejecución. Si bien existen herramientas que contemplan la simulación de algunas de éstas situaciones (por ejemplo, fallos en computadores y red) [7, 8, 9], es difícil diseñar, generar y ejecutar las pruebas de forma determinista ya que hay simular en detalle gran cantidad de los elementos que componen la infraestructura, incluyendo el propio framework.

En este trabajo se presenta una técnica que permite generar automáticamente las diferentes configuraciones de ejecución para una aplicación *MapReduce* en un entorno de desarrollo/pruebas. A partir de los datos de entrada de las pruebas, se obtienen las configuraciones basándose en las diferentes ejecuciones que pueden ocurrir en producción. Cada una de estas configuraciones se ejecuta posteriormente en un entorno de pruebas con el objetivo de reproducir los defectos funcionales del programa que podrían ocurrir en producción. Las contribuciones de este trabajo son:

1. Técnica combinatoria que, a partir de unos datos de prueba, genera las diferentes configuraciones de infraestructura que podrían producirse en un entorno de producción teniendo en cuenta las características de procesamiento de *MapReduce*.
2. Soporte automático para lo anterior con una extensión de MRUnit [10], de forma que permite generar las configuraciones de infraestructura, su ejecución y evaluar si se ha producido un fallo.

El resto del artículo es como sigue. En la sección 2 se resumen los fundamentos del paradigma *MapReduce*. La generación de las diferentes configuraciones, así como la ejecución y automatización de las pruebas se define en la sección 3. En la sección 4 se aplica a un caso de estudio. En la sección 5 se discute el trabajo relacionado sobre las pruebas en los programas *MapReduce*, y finalmente las conclusiones y el trabajo futuro en la sección 6.

2 Paradigma MapReduce

Los programas *MapReduce* procesan grandes cantidades de datos en infraestructuras distribuidas. Para ello, el desarrollador crea dos funcionalidades: Mapper que divide

el problema en varios subproblemas, y Reducer que resuelve estos subproblemas. El resultado final se obtiene a partir del despliegue y ejecución controlada de varias instancias Mapper y Reducer denominadas tarea. Esta labor se realiza automáticamente por *Hadoop* u otro framework. En primer lugar se ejecutan en paralelo varias tareas Mapper que analizan un subconjunto de los datos de entrada y determinan qué subproblemas necesitan esa información. Cuando todas las Mapper finalizan, se ejecutan también en paralelo varias tareas Reducer para resolver los subproblemas. Internamente *MapReduce* maneja pares <clave, valor>, donde la clave es el identificador del subproblema y el valor contiene la información que se necesita para resolverlo.

Suponer a modo de ejemplo que se tiene una gran cantidad de datos históricos sobre temperaturas y que se realiza un programa *MapReduce* para calcular la temperatura media por año. Este programa resuelve un subproblema por cada año, por lo que el identificador del subproblema o clave es el año. La tarea Mapper recibe un subconjunto de datos de las temperaturas y emite pares <año, temperatura de ese año>. A continuación, *Hadoop* agrupa todos los valores por su clave. Por tanto, a la tarea Reducer le llegan subproblemas del tipo <año, [todas las temperaturas de ese año]>, es decir, por cada año se tienen agrupadas todas las temperaturas y finalmente calcula la media. Por ejemplo, en la figura 1 se detalla una ejecución del programa considerando como entrada: año 2000 con 3º, 2002 con 4º, 2000 con 1º, y 2001 con 5º. Las dos primeras entradas se analizan en una tarea Mapper y el resto en otra. A continuación, por cada año se agrupan todas sus temperaturas y se envían a una tarea Reducer. La primera Reducer recibe todas las temperaturas de los años 2000 y 2002, y la otra las del año 2001. Finalmente cada una emite la temperatura media de los subproblemas que analiza: 2º en el 2000, 4º en el 2002, y 5º en el 2001. Este programa con la misma entrada podría ser ejecutado por el framework de distinta forma, por ejemplo con tres Mapper y tres Reducer. Independientemente de cómo se ejecute debería generar la salida esperada.

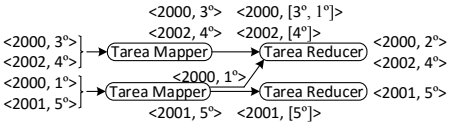


Fig. 1. Ejecución de programa que calcula la temperatura media por año

Adicionalmente, para optimizar el programa se puede implementar la funcionalidad Combiner que se ejecuta después de la tarea Mapper y elimina aquellos pares <clave, valor> que son irrelevantes para resolver el subproblema. También existen otras funcionalidades, como por ejemplo Partitioner que decide a qué tarea Reducer se envía cada par <clave, valor>, Sort que ordena los pares <clave, valor> o Group que determina cómo se agrupan los valores por cada clave antes de llegar a Reducer.

La incorrecta implementación de estas funcionalidades podría causar fallos en alguna de las diferentes formas en que *Hadoop* ejecuta el programa. Estos defectos son

difíciles de detectar en las pruebas ya que debido a que se tienen pocos datos se suele ejecutar sólo una Mapper, luego una Combiner y finalmente una Reducer.

3 Generación y Ejecución de Pruebas

En esta sección se define la generación de configuraciones de prueba (sección 3.1), y un marco general sobre el que ejecutarlas (sección 3.2).

3.1 Generación de escenarios de prueba

Con el objetivo de mostrar cómo las configuraciones de la infraestructura pueden afectar al resultado de los programas, se considera el ejemplo de la sección 2 pero al que se le añade erróneamente una tarea Combiner con el objetivo de reducir los datos que se envían por la red y así mejorar el rendimiento. La tarea Combiner recibe varias temperaturas y las sustituye por un único dato que contiene su media. Al añadirle la tarea Combiner se introduce un defecto funcional ya que Reducer necesita calcular la temperatura media de un año y no la puede obtener con las temperaturas medias parciales que le llegan de Combiner. En la figura 2 se muestran tres ejecuciones que puede seguir el programa en producción considerando diferentes configuraciones de la infraestructura ante un misma entrada.

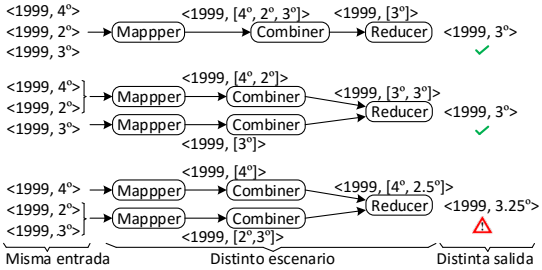


Fig. 2. Ejecución de pruebas para un programa que calcula la temperatura media por año

La primera configuración consiste en una Mapper, una Combiner y una Reducer que producen la salida esperada. La segunda configuración también genera la salida esperada ejecutando una Mapper que procesa las temperaturas 4º y 2º, otra Mapper para 3º, dos Combiner, y finalmente una Reducer. La tercera configuración también ejecuta dos Mapper, dos Combiner y una Reducer, pero produce una salida no esperada debido a que la primera Mapper procesa 4º y la segunda las temperaturas 2º y 3º. Luego una de las tareas Combiner hace la media de 4º, y la otra de las temperaturas 2º y 3º, de forma que a Reducer le llegan 4º y 2.5º. Finalmente la tercera configuración

obtiene que la temperatura media es de 3.25º en lugar de 3º, revelando el defecto funcional. Siempre que se ejecute esta configuración de la infraestructura se detecta el defecto, independientemente que ocurran fallos de computadores, de red u otros. En cambio, se enmascara al ejecutarse otra configuración. El fallo anterior es difícil de detectar ya que se tiene que conocer qué configuración de la infraestructura lo revela y ejecutarla de forma totalmente controlada.

El objetivo es que a partir de unos datos de prueba se generen las diferentes configuraciones de infraestructura, también denominadas *escenarios*. Para ello se tiene en cuenta cómo se podrían ejecutar esos datos de prueba en producción. Los programas *MapReduce* primero ejecutan las Mapper, sobre su salida se ejecutan las Combiner y finalmente las Reducer. Cada ejecución puede realizarse sobre distinto número de computadores y por tanto las tareas Mapper/Combiner/Reducer procesan diferentes conjuntos de datos en cada ejecución. Para generar cada uno de los *escenarios* se propone una técnica que combina [11] los valores de los diferentes parámetros que pueden modificar la ejecución del programa *MapReduce* y producir fallos de acuerdo a la clasificación de defectos MRTree [5]. Estos parámetros son:

- Parámetros Mapper, para los datos de entrada:
  1. Número de tareas Mapper.
  2. Las entradas que procesa cada Mapper.
  3. Orden de procesamiento de los datos de entrada, es decir, qué datos se procesan antes y cuáles después.
- Parámetros Combiner, para la salida de cada tarea Mapper:
  4. Número de tareas Combiner.
  5. Las entradas que procesa cada Combiner.
- Parámetros Reducer, para los datos que le llegan de Mapper y Combiner:
  6. Número de tareas Reducer.
  7. Las entradas que procesa cada Reducer.

Los diferentes *escenarios* se obtienen combinando todos los valores que pueden tomar los parámetros y aplicando las restricciones que impone la ejecución secuencial de tareas *MapReduce*:

1. Los valores/combinaciones de los parámetros de Mapper dependen de los datos de entrada ya que no pueden existir más tareas que datos.
2. Los de Combiner dependen del resultado de las tareas Mapper
3. Los de Reducer dependen del resultado de las tareas Combiner.

Para ilustrar la combinatoria de parámetros y sus restricciones se utiliza como ejemplo el programa de la figura 2. Este tiene una entrada con tres datos, por lo que estos datos restringen los valores que pueden tomar los parámetros de Mapper ya que como máximo podría tener 3 Mapper en paralelo (cada una analizando un dato). El primer *escenario* se genera con una Mapper, una Combiner y una Reducer. Para el segundo *escenario* se modifica el parámetro "Número de tareas Mapper" a 2 donde la primera analiza dos pares <clave, valor> y la segunda uno. El tercer escenario mantiene el parámetro "Número de tareas Mapper" en 2, pero modifica el parámetro "Las entra-

das que procesa cada Mapper" de forma que la primer Mapper analiza un par <clave, valor> y la segunda dos. Modificando de esta forma los valores que pueden tomar los parámetros se van generando los diferentes *escenarios* teniendo en cuenta las restricciones.

3.2 Ejecución de escenarios de prueba

En la sección anterior se ha propuesto una técnica para generar *escenarios* que representan diferentes configuraciones de la infraestructura teniendo en cuenta características del procesamiento *MapReduce*, A continuación, se describe en la figura 3 un marco para ejecutar sistemáticamente las pruebas bajo los diferentes *escenarios* generados con la técnica de la sección anterior.

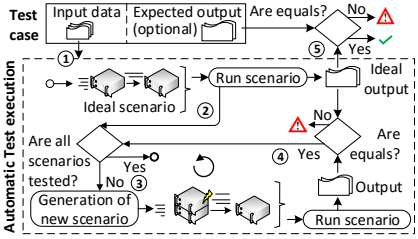


Fig. 3. Marco de ejecución de pruebas

Se parte de un caso de prueba que tiene datos de entrada y opcionalmente la salida esperada (1). Esos datos de entrada se pueden obtener previamente con diferentes técnicas de pruebas genéricas o diseñadas específicamente para *MapReduce* como por ejemplo MRFlow [12]. Se comienza ejecutando los datos de entrada en el *escenario ideal*, que es aquel formado por una configuración con una Mapper, una Combiner y una Reducer, tal y como habitualmente se ejecutan las pruebas (2). A continuación y de forma iterativa se generan y ejecutan nuevos *escenarios* mediante la técnica de la sección anterior (3). Se comprueba que la salida de cada *escenario* sea equivalente a la salida del *escenario ideal*, en caso contrario se ha revelado un defecto a pesar de desconocer cuál es la salida esperada (4). Finalmente, si en el caso de prueba se especifica la salida esperada, se comprueba que también sea equivalente a la del *escenario ideal* (5), sino se detecta un defecto.

Partiendo del caso de prueba, se generan los *escenarios* según la técnica de la sección anterior, y luego se ejecutan y evalúan iterativamente tal y como se describe en el siguiente pseudocódigo:

```
Entrada: caso de prueba formado por:
  (1) datos de entrada
  (2) salida esperada (opcional)
```

```
Salida: escenario que revela el defecto
0  /* Generación de escenarios (sección 3.1) */
1  Escenarios ← Generar escenarios (datos de entrada)
2  /* Ejecución de escenarios */
3  salida escenario ideal ← Ejecución escenario ideal
4  ∀ escenario ∈ Escenarios:
5     salida escenario ← ejecución de escenario
6     SI salida escenario <> salida escenario ideal:
7       EMITIR escenario con fallo
8  SI salida escenario ideal <> salida esperada:
9    EMITIR escenario ideal
10 SINO no se ha revelado ningún defecto
```

Por ejemplo, en la figura 2 se muestra la generación y ejecución de 3 *escenarios* para un caso de prueba. Primero se ejecuta el *escenario ideal* con una Mapper, una Combiner y una Reducer que produce 3º como salida. Luego se ejecuta el segundo *escenario* que también produce 3º. Finalmente se ejecuta un tercer *escenario* que produce como salida 3.25º que no es equivalente a los 3º de la salida del *escenario ideal*. Por tanto se revela un defecto funcional sin conocer la salida esperada del caso de prueba.

El anterior enfoque se automatiza en un motor de ejecución de pruebas a partir de una extensión de la herramienta Apache MRUnit [10]. Ésta sólo ejecuta el *escenario ideal*, por lo que se ha modificado para generar el resto de *escenarios* y ejecutar varias tareas Mapper, Combiner y Reducer.

4 Caso de Estudio

Con el objetivo de evaluar la técnica de prueba, ésta se aplica a Open Ankus [13], una herramienta de minería de datos y aprendizaje automático implementada en *MapReduce*. Consiste en un sistema de recomendación, el cual predice y recomienda varios elementos (libros, películas, etc.) a cada usuario basándose en los gustos almacenados en su perfil. Una funcionalidad del programa se encarga de comprobar el grado de acierto de las recomendaciones, mediante la comparación de la puntuación que el sistema predijo y la que proporcionó el usuario. Esta funcionalidad tiene un diseño MultipleInputs que consiste en dos implementaciones distintas de tareas Mapper: una Mapper recibe archivos con las predicciones del sistema y otra con las puntuaciones del usuario, pero ambas emiten datos a una única implementación de Reducer. Las tareas Mapper reciben las predicciones y puntuaciones de los usuarios sobre los diferentes elementos y las agrupan para cada par usuario-elemento. Las tareas Reducer reciben para cada par usuario-elemento todas las predicciones del sistema y las puntuaciones que fueron proporcionadas por el usuario, por lo que finalmente calcula lo acertadas que fueron tales predicciones.

Sobre el programa anterior, se obtiene un caso de prueba utilizando un análisis de flujo de datos específico para *MapReduce* [12]. El caso de prueba tiene como datos de entrada las siguientes dos predicciones junto con sus puntuaciones: (1) el sistema predice que Laura puntuará El Quijote con 0 puntos, (2) Laura puntúa El Quijote con

0 puntos, (3) posteriormente el sistema detecta que los gustos de Laura han cambiado y predice que puntuará El Quijote con 10 puntos, y (4) Laura puntúa con 10 puntos El Quijote. Estos datos de entrada se proporcionan en dos ficheros, uno de puntuaciones y otro de predicciones. La salida esperada es que el sistema acertó al 100% sus predicciones.

Partiendo del caso de prueba anterior, se ha aplicado el procedimiento descrito en la sección 3. Se han generado y ejecutado 49 *escenarios* de pruebas, detectando un defecto que se produce en 23 *escenarios* (47%). Este defecto sólo se revela cuando unas entradas se procesan antes que otras como ocurre en producción con grandes cantidades de datos, de forma que el programa no es capaz de asignar adecuadamente cada predicción con su puntuación real. Los *escenarios* que revelan el defecto asocian la predicción 10 con la puntuación 0, y la predicción 0 con la puntuación 10, tal y como se representa en la parte inferior de la figura 4. Este *escenario* parte de una Mapper que procesa las predicciones y otras dos Mapper que procesan respectivamente las puntuaciones 0 y 10. Además, este *escenario* fuerza a que se procese primero la puntuación 10 que la puntuación 0, llegando a la tarea Reducer que el par Laura-El Quijote tiene asignadas dos predicciones de 0 y 10 puntos, y que hay dos puntuaciones de 10 y 0 puntos. La tarea Reducer asocia la primera predicción con la primera puntuación y así sucesivamente, por lo que obtiene incorrectamente como salida que el sistema no acierta en las predicciones.

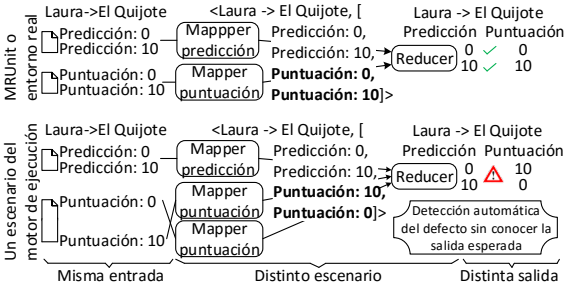


Fig. 4. Ejecución del caso de prueba en distintos escenarios

En cambio, el caso de prueba no revela el defecto si lo ejecutamos en los siguientes entornos: (a) cluster *Hadoop* de producción con 4 computadores, (b) *Hadoop* en modo local (versión más simple de *Hadoop* con un único computador), y (c) librería de pruebas unitarias MRUnit. En la tabla 1 se resumen sus resultados acompañados del tiempo de ejecución del caso de prueba. Estos entornos, a diferencia del motor de ejecución propuesto en este artículo, enmascaran el defecto al ejecutar el caso de prueba en un único *escenario*. Este se corresponde con el *escenario ideal* representa-

do en la parte superior de la figura 4, donde sólo hay dos Mapper, una para las predicciones y otra para las puntuaciones, y una Reducer.

Tabla 1. Resultados del caso de estudio

Evaluación\entorno	Cluster producción	Hadoop local	MRUnit	Motor propuesto
Nº defectos	0	0	0	1
Tiempo ejecución	2 min. 47 s. 67 ms.	4s. 19 ms.	437 ms.	468 ms.

El motor de ejecución de pruebas propuesto en este artículo ejecuta los casos de prueba en los diferentes *escenarios* que pueden ocurrir en producción. Además, a diferencia del resto de entornos analizados, no necesita la salida esperada para detectar defectos funcionales. Por ejemplo, este caso de estudio revela el defecto al comprobar que la salida del *escenario ideal* (el sistema acierta con las predicciones) no es equivalente a la de otro *escenario* (el sistema no acierta con las predicciones).

5 Trabajo Relacionado

A pesar de los desafíos de las pruebas sobre aplicaciones en Big Data [14, 15] y de los avances de técnicas relacionadas con éstas [16], se han realizado pocos esfuerzos orientados a la prueba de aplicaciones *MapReduce* [17] pese a ser uno de los principales paradigmas de procesamiento utilizados Big Data [18]. En un estudio de Kavulya et al. [19] donde se analizan varios programas *MapReduce*, el 3% no terminan de ejecutarse, mientras que otro estudio de Ren et al. [20] lo sitúa entre el 1.38% y el 33.11%.

La mayoría de investigaciones existentes de pruebas para aplicaciones *MapReduce* se centran en el rendimiento y en menor medida en la funcionalidad. En un enfoque de pruebas en programas Big Data propuesto por Gudipati et al. [21] se especifica un proceso específico para la validación en *MapReduce*. Dentro de este, Camargo et al. [22] y Morán et al. [5] identifican y clasifican varios defectos funcionales. Algunos de estos defectos son específicos del paradigma *MapReduce* y no son fáciles de detectar ya que dependen de la ejecución sobre la infraestructura. Un tipo de defecto muy común es el que se produce cuando se espera que los datos lleguen a Reducer en un determinado orden, pero a causa de la ejecución paralela llegan desordenados. Este defecto ha sido analizado por Csallner et at. [23] que proponen detectarlo con un enfoque basado en ejecución simbólica y Chen et al. [24] basándose en model checking. A diferencia de los anteriores, el enfoque presentado en este trabajo no solamente está dirigido a detectar un tipo de defecto, sino que permite detectar otros específicos de *MapReduce*. Para ello, se ejecutan los datos de las pruebas con distintas configuraciones de infraestructura.

Varias investigaciones sugieren realizar pruebas con fallos en la infraestructura [25, 26], existiendo varias herramientas que soportan la inyección de los fallos [7, 8, 9]. Por ejemplo, las investigaciones de Marynowski et al. [27] permiten crear casos de prueba especificando qué computadores fallan y en qué momento. Uno de los posibles

problemas es que pueden existir defectos específicos de *MapReduce* que no se detectan con fallos de computadores, sino que requieren un control total de *Hadoop* y de la infraestructura. En este trabajo se generan automáticamente las distintas formas en las que *Hadoop* podría ejecutar el programa desde el punto de vista funcional, independientemente de que se produzcan por fallos de la infraestructura u optimizaciones de *Hadoop*.

Por otra parte, existen enfoques orientados a obtener las entradas de prueba para aplicaciones *MapReduce*, como [12] donde se utiliza análisis de flujo de datos y [28] basadas en algoritmos bacteriológicos. En este trabajo se generan las configuraciones sobre las que ejecutar unos datos de prueba dados. Estos datos de prueba podrían obtenerse de las anteriores técnicas de prueba.

6 Conclusiones y Trabajo Futuro

En este artículo se ha elaborado una técnica de pruebas y se ha automatizado en un motor de ejecución de pruebas para programas implementados según el modelo *MapReduce*. Éste motor reproduce para un caso de prueba dado las ejecuciones causadas por las diferentes configuraciones de la infraestructura. Automáticamente, y sin necesidad de conocer la salida esperada, puede detectar defectos funcionales específicos del paradigma *MapReduce* que en muchos casos son difíciles de detectar en entornos de prueba y producción. Se ha aplicado a un programa que actualmente está en producción, y para un caso de prueba con pocos datos ha generado y ejecutado automáticamente 49 configuraciones de infraestructura distintas, de las cuales 23 (47%) revelan un defecto.

Como trabajo futuro se planifica extender la técnica para seleccionar eficientemente aquellas configuraciones con más probabilidad de hacer que el programa falle. El enfoque actual es *off-line* ya que las pruebas no se realizan cuando el programa está en producción. Se plantea extender el actual enfoque para realizar pruebas *on-line* monitorizando la funcionalidad con los datos reales de producción y detectando automáticamente los defectos.

Agradecimientos

Este trabajo ha sido realizado bajo el proyecto de investigación TIN2013-46928-C3-1-R, financiado por el Ministerio de Economía y Competitividad, y fondos FEDER. También ha sido realizado bajo el proyecto GRUPIN14-007, financiado por el Principado de Asturias y fondos FEDER.

Referencias

1. ISO/IEC JTC 1 – big data, preliminary report 2014 (2015)

2. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: Proc. of the OSDI - Symp. on Operating Systems Design and Implementation, USENIX (2004) 137–149
3. Apache Hadoop: open-source software for reliable, scalable, distributed computing. https://hadoop.apache.org Accessed: 2016-04-16.
4. Instituciones que utilizan Hadoop con fines educativos o de producción. http://-wiki.apache.org/hadoop/PoweredBy Accessed: 2016-04-16.
5. Morán, J., de la Riva, C., Tuya, J.: MRTree: Functional Testing Based on MapReduce's Execution Behaviour. In: Future Internet of Things and Cloud (FiCloud), 2014 International Conference on. (2014) 379–384
6. Vishwanath, K.V., Nagappan, N.: Characterizing cloud computing hardware reliability. In: Proceedings of the 1st ACM symposium on Cloud computing, ACM (2010) 193–204
7. Hadoop Injection Framework. https://hadoop.apache.org Accessed: 2016-04-16.
8. Chaos Monkey. https://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey Accessed: 2016-04-16.
9. AnarchyApe: Fault injection tool for Hadoop cluster from Yahoo anarchyape. https://-github.com/david78k/anarchyape Accessed: 2016-04-16.
10. Apache MRUnit: Java library that helps developers unit test Apache Hadoop map reduce jobs. http://mrunit.apache.org Accessed: 2016-04-16.
11. Grindal, M., Offutt, J., Andler, S.F.: Combination testing strategies: a survey. Software Testing, Verification and Reliability **15**(3) (2005) 167–199
12. Morán, J., de la Riva, C., Tuya, J.: Testing Data Transformations in MapReduce Programs. In: Proceedings of the 6th International Workshop on Automating Test Case Design, Selection and Evaluation. A-TEST 2015, New York, NY, USA, ACM (2015) 20–25
13. Open Ankus: Data mining and machine learning based on MapReduce. http://-www.openankus.org/ Accessed: 2016-04-16.
14. Nachiyappan, S., Justus, S.: Getting ready for bigdata testing: A practitioner's perception. In: Computing, Communications and Networking Technologies (ICCCNT), 2013 Fourth International Conference on, IEEE (2013) 1–5
15. [15] Mittal, A.: Trustworthiness of big data. International Journal of Computer Applications **80**(9) (2013)
16. Bertolino, A.: Software testing research: Achievements, challenges, dreams. In: Future of Software Engineering, 2007. FOSE '07. (2007) 85–103
17. Camargo, L.C., Vergilio, S.R.: Mapreduce program testing: a systematic mapping study. In: Chilean Computer Science Society (SCCC), 32nd International Conference of the Computation. (2013)
18. Sharma, M., Hasteer, N., Tuli, A., Bansal, A.: Investigating the inclinations of research and practices in hadoop: A systematic review Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference -.
19. Kavulya, S., Tan, J., Gandhi, R., Narasimhan, P.: An analysis of traces from a production mapreduce cluster. In: Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, IEEE (2010) 94–103
20. Ren, K., Kwon, Y., Balazinska, M., Howe, B.: Hadoop's adolescence: an analysis of hadoop usage in scientific workloads. Proceedings of the VLDB Endowment **6**(10) (2013) 853–864
21. Gudipati, M., Rao, S., Mohan, N.D., Gajja, N.K.: Big data: Testing approach to overcome quality challenges. Big Data: Challenges and Opportunities (2013) 65–72

22. Camargo, L.C., Vergilio, S.R.: Cassicação de defeitos para programas mapreduce: resultados de um estudo empírico. In: SAST - 7th Brazilian Workshop on Systematic and Automated Software Testing. (2013)
23. Csallner, C., Fegaras, L., Li, C.: New ideas track: testing mapreduce-style programs. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ACM (2011) 504–507
24. Chen, Y.F., Hong, C.D., Sinha, N., Wang, B.Y.: Commutativity of reducers. In: Tools and Algorithms for the Construction and Analysis of Systems. Springer (2015) 131–146
25. Faghri, F., Bazarbayev, S., Overholt, M., Farivar, R., Campbell, R.H., Sanders, W.H.: Failure scenario as a service (fsaas) for hadoop clusters. In: Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management, ACM (2012) 5
26. Joshi, P., Gunawi, H.S., Sen, K.: Prefail: A programmable tool for multiple-failure injection. In: ACM SIGPLAN Notices. Volume 46., ACM (2011) 171–188
27. Marynowski, J.E., Santin, A.O., Pimentel, A.R.: Method for testing the fault tolerance of mapreduce frameworks. Computer Networks **86** (2015) 1–13
28. Mattos, A.J.: Test data generation for testing mapreduce systems. In: Master's degree dissertation. (2011)

Sobre el grado de acuerdo entre evaluadores en la detección de *Design Smells*

Khalid AlKharabsheh[*1], Yania Crespo[2], Esperanza Manso[2], and José Ángel Taboada[1]

[1]{khalid,joseangel.taboada}@usc.es, CiTIUS
Centro Singular de Investigación en Tecnologías de la Información
Universidad de Santiago de Compostela, Santiago de Compostela 15782
[2]{yania,manso}@infor.uva.es, Dpto. Informática, Universidad de Valladolid
Escuela de Ingeniería Informática. Campus Miguel Delibes, Valladolid 47011

Resumen La detección automática de *Design Smells* ha evolucionado a la par que las herramientas automáticas de *refactoring*. Sin embargo se constata que su grado de adopción por la industria de desarrollo del software no ha sido comparable con la forma en la que se han adoptado las herramientas de *refactoring*. En este trabajo partimos de la suposición de que la diferencia reside en la objetividad y pragmatismo de una operación de *refactoring* comparada con el grado de subjetividad que suponemos en la definición e identificación de *Design Smells*. Para estudiar este problema se diseñó una encuesta difundida vía correo electrónico en la que se obtuvieron 92 respuestas de personas tanto de la academia como de la industria acerca de la presencia de *Design Smells* en 5 clases de un proyecto de código abierto. El estudio se ha realizado centrándose en la detección de dos tipos de *Design Smells*: *God Class* y *Feature Envy*. Se ha realizado un análisis en el que se valora el grado de acuerdo en la identificación de estos *Design Smells* entre los encuestados mediante el estadístico *Kappa-Fleiss*. En el trabajo se analizan la interrelación entre diferentes factores como la experiencia del evaluador, su contexto de trabajo, etc. Los resultados obtenidos muestran que no existe acuerdo en general o que es muy pobre en los casos que sí existe algún acuerdo. Estos resultados sostienen que, por una parte, falta formación en *Design Smells* y, por otra parte, hay una componente subjetiva que hace a distintos sujetos evaluar de forma distinta la presencia o no del *Design Smell*.

Keywords: Detección de *Design Smells*, encuesta, acuerdo entre evaluadores, *Kappa-Fleiss*, calidad, evolución, mantenimiento

1. Introducción

Un *Design Smell* es un problema encontrado en la estructura del software (código, diseño) que no produce errores de compilación o de ejecución, pero afecta negativamente los factores de calidad del software. *Design Smell* es un término unificador que agrupa en un único concepto a *Code Smells*, *Bad Smells*,

[*] Khalid AlKharabsheh agradece la financiación del programa Erasmus Mundus

Disharmonies, *Antipatterns*, *Design flaws*, *Technical debt*, entre otros términos conocidos en el estado del arte.

La detección de *Design Smells* ha ido experimentado un auge en resultados de investigación publicados. La primera definición de *Design Smells* data del año 2000. La primera herramienta se reporta en 2002. A partir de 2004 comienza un crecimiento continuado, proliferando la aparición de nuevas herramientas para la detección automática de *Design Smells*. Particularmente a partir de 2009-2010 se produce un incremento notable en la actividad en este ámbito.

Nos encontramos actualmente con todo tipo de herramientas para detectar *Design Smells*: herramientas dedicadas, integradas como plug-ins en entornos de desarrollo, aplicando técnicas diferentes, para varios lenguajes de programación, etc.. Sin embargo, a pesar de este auge, no se ha experimentado una adopción de estas herramientas por parte de la industria. La detección automática de *Design Smells* ha evolucionado a la par que las herramientas automáticas de *refactoring*. Teniendo en cuenta que la presencia de *Design Smells* sirve como base para la detección de oportunidades de textitrefactoring, es llamativo que su grado de adopción por parte de los desarrolladores en la producción y mantenimiento del software no sea comparable con la forma en la que se han adoptado las herramientas de *refactoring*. En este trabajo partimos de la suposición de que la diferencia reside en la objetividad y pragmatismo de una operación de *refactoring* comparada con el grado de subjetividad que suponemos en la definición e identificación de *Design Smells*.

Partiendo de estudios comparativos sobre herramientas de detección de *Design Smells*, se aprecia que no hay buen acuerdo entre ellas a la hora de detectar la presencia de un *Design Smell*. En trabajos previos [5], nuestro grupo ha explorado la cuestión de modelar la subjetividad en la detección de *Design Smells*, introduciendo información semántica sobre la intención de diseño del desarrollador, así como teniendo en cuenta históricos de datos de las organizaciones y la retroalimentación de falsos negativos y falsos positivos detectados por expertos. Estas ideas relativas a lo subjetivo de la detección de *Design Smells* nos conducen a la necesidad de realizar un estudio para comprobar cómo coinciden o no las personas, desarrolladores e investigadores al decidir la presencia de un *Design Smell* en el software. En consecuencia, se elaboró el experimento que aquí se presenta.

Para estudiar este problema se diseñó una encuesta difundida vía correo electrónico en la que se obtuvieron 92 respuestas de personas tanto de la academia como de la industria acerca de la presencia de *Design Smells* en 5 clases de un proyecto de código abierto. El estudio se ha realizado centrándose en la detección de dos tipos de *Design Smells*: *God Class* y *Feature Envy*. La selección se basa en que son dos de los *Design Smells* más populares en la literatura y en que además se trata de *Design Smells* de diferente naturaleza en cuanto al ámbito y al efecto en el software. Se ha realizado un análisis en el que se valora el grado de acuerdo en la identificación de *Design Smells* entre los encuestados mediante el estadístico *Kappa-Fleiss* [3]. En el trabajo se analizan la interrelación entre diferentes factores como la experiencia del evaluador, su contexto de trabajo, etc. Los resultados obtenidos muestran que no existe acuerdo en general y que es muy pobre en los casos en los que sí existe algún acuerdo. A partir de los resultados obtenidos se concluye con recomendaciones a tener en cuenta en

las nuevas tendencias para la detección automática de *Design Smells* que pueden influir positivamente en la adopción por la industria de técnicas y herramientas para la detección de *Design Smells*.

El resto del artículo se organiza de la siguiente forma. En la Sección 2 se describe el marco conceptual para este trabajo. Se tratan cuestiones relativas a la detección de los defectos de diseño que nos ocupan. En la Sección 3 se detallan los objetivos del trabajo y las preguntas de investigación que nos planteamos responder. Se describe el diseño de la encuesta realizada de forma que permita obtener datos para responder a las preguntas de investigación. La Sección 4 presenta los estudios realizados a partir de las respuestas de los evaluadores, los resultados obtenidos y el análisis de los mismos. En la Sección 5 se describen brevemente algunos trabajos relacionados mientras que la Sección 6 presenta un resumen de las conclusiones obtenidas y las líneas de trabajo a partir del mismo.

2. Detección de los *Design Smells God Class* y *Feature Envy*

El estudio se ha realizado centrándose en la detección de dos tipos de *Design Smells*, *God Class* y *Feature Envy*. Se trata de dos tipos diferentes. *God Class* es un *Design Smell* intra clase, sólo se necesita observar el código de dicha clase para detectarlo. Por su parte *Feature Envy* es un *Design Smell* inter clase, para detectarlo es relevante observar la interacción de dicha clase con otras clases relacionadas.

En los buenos diseños orientados a objetos la lógica del sistema está distribuida uniformemente entre las clases de los niveles superiores. En [9] se define una *God Class* como un objeto que controla demasiados objetos en el sistema y que ha crecido más allá de toda lógica para convertirse en la clase que lo hace todo. En una *God Class* hay demasiadas variables de instancia y demasiado código. Donde hay demasiado código hay peligro de que surja código duplicado. A veces se corresponde con una mala aplicación del patrón de diseño Mediador. Este problema de diseño puede ser parcialmente asimilado con el defecto *Large Class* definido por Fowler [4]. También es conocido como el antipatrón *"The Blob"* en [2].

Feature Envy se recoge en el catálogo de Fowler [4] y se clasifica por Wake [11] en la categoría que denomina "entre clases" y la subcategoría "de responsabilidad". Un método tiene el defecto *Feature Envy* si parece estar más preocupado en manipular datos de otras clases que de la suya propia. Está relacionado con la responsabilidad de la clase, ya que contiene métodos que deberían estar en otra clase. Como excepciones Fowler indica que existen varios patrones de diseño que rompen esta regla, como el Visitante y el Estrategia.

Se trata entonces de dos *Design Smells* de ámbito diferente. *God Class* es de ámbito de clase. Para detectarlo hay que revisar la clase como un todo. *Feature Envy* por su parte es de ámbito de método, para detectarlo basta con revisar en el interior de un método. De acuerdo con Tiberghien et al. [10] en su clasificación de *Design Smells*, *God Class* pertenece al grupo de los *"Bloaters"* (hinchadores) mientras que *Feature Envy* pertenece al grupo de los *"Couplers"* (acopladores).

3. Objetivos y preguntas de investigación

El estudio previo que hemos realizado, sobre el grado de acuerdo entre herramientas al detectar *Design Smell* en una colección de clases Java [1], nos ha dado pie para profundizar en el escenario de detección de *Design Smell* por personas (evaluadores), estudiando el papel que juega la subjetividad en dicha detección.

Utilizando el patrón **GQM**, ampliamente extendido en la investigación en ingeniería del software para definir los objetivos [12], el objetivo de este estudio cuasi experimental se define como:

Analizar Una colección de clases
Con el propósito de evaluarlas
Con respecto al grado de acuerdo entre evaluadores para detectar los *Design Smells God Class* y *Feature Envy*
Desde el punto de vista de profesores, investigadores y desarrolladores de software
En el contexto académico de los investigadores que realizan el estudio

De este objetivo general derivamos la siguiente hipótesis de trabajo, y cuestiones a contestar.

Hipótesis 1 *No existe acuerdo entre los evaluadores en la detección de los Design Smells God Class y Feature Envy*
Esta hipótesis general se desglosa en las siguientes hipótesis secundarias:

Hipótesis 1.a *No existe acuerdo entre los evaluadores humanos en la detección de Design Smells.*

Hipótesis 1.b *No existe acuerdo entre evaluadores humanos y herramientas en la detección de Design Smells.*

Queremos conocer qué factores pueden afectar el acuerdo o el desacuerdo en la detección de *Design Smells*, qué interrelación puede existir entre estos factores y qué impacto tiene esto en detección. Pensamos que puede haber factores que afecten de alguna forma a la detección de *Design Smells* cuando se trata de evaluadores humanos, tales como: el grado de experiencia de los desarrolladores, sus antecedentes (en términos de formación, conocimientos y actividades desarrolladas), el contexto de trabajo, incluso el área geográfica en la que se han formado y/o trabajan.

Para estudiar este problema se diseñó una encuesta online, cuyo objetivo principal era permitir la detección de los *Design Smells God Class* y *Feature Envy* por sujetos (evaluadores humanos). Las preguntas de investigación que nos planteamos, y que han dirigido tanto el diseño de la encuesta como la explotación de los resultados, son las siguientes:

RQ1 ¿Existe acuerdo entre los evaluadores humanos al detectar *Design Smells*?
RQ2 ¿Existe acuerdo entre los evaluadores humanos y las herramientas de detección de *Design Smells*?
RQ3 ¿Qué herramientas coincide más con los evaluadores humanos al detectar *Design Smells*?

RQ4 ¿Cómo afecta el grado de experiencia en la detección de *Design Smells*?
- **RQ4a** ¿El grado de acuerdo entre evaluadores humanos es mayor en el grupo de evaluadores con mayor experiencia?
- **RQ4b** ¿El grado de acuerdo con las herramientas de detección es mayor cuando el evaluador humano tiene mayor experiencia?

RQ5 ¿Cómo afectan los antecedentes (en términos de formación, experiencia y conocimientos) y el contexto de los evaluadores en la detección de *Design Smells*?
- **RQ5a** ¿Afecta el contexto de trabajo? ¿el área geográfica donde se forma o desarrolla la persona? ¿si se trata de un contexto académico o de la industria?
- **RQ5b** ¿Afectan los antecedentes del evaluador? ¿experiencia en programación orientada a objetos, en lecturas o revisiones de código? ¿nivel de conocimiento de *Design Smells*?

3.1. Diseño del experimento

La encuesta y las clases a evaluar como candidatas a presentar *Design Smells*, se han elegido para que los sujetos no necesiten demasiado tiempo para completar las tareas. Puesto que la tarea de detección de *Design Smells* es consumidora de tiempo, solamente se van a proporcionar a los encuestados 5 clases que serán evaluadas por los sujetos para identificar en ellas posibles *Design Smells*. Como se ha mencionado anteriormente, se preguntará por la detección de *God Class* y *Feature Envy* en las clases seleccionadas. La elección de *God Class* y *Feature Envy* para el estudio es interesante ya que se trata de dos tipos de *Design Smell* diferentes como se explicó en la sección 2, lo que podría ser útil a la hora de extraer conclusiones de los resultados obtenidos.

Para facilitar la tarea del evaluador se suministrará a los encuestados una forma rápida de recordar la definición de dichos *Design Smells*. En el diseño de la encuesta, además de las preguntas directamente relacionadas con la detección de los DS, hay otra colección de preguntas que tiene como objetivo elaborar el perfil del sujeto encuestado, con el fin de comprobar si los factores del perfil influyen en el acuerdo en la detección de *Design Smells*, tal y como hemos explicado en las cuestiones de investigación.

La selección de las 5 clases debe cumplir los siguientes requisitos:

- se debe poder leer la clase separadamente y a la vez permitir ver la clase en su contexto para ayudar al evaluador en su tarea.
- las clases elegidas deben ser un subconjunto de las clases que se utilizan en el primer experimento de comparación entre herramientas que se presenta en [1].
- en el criterio de elección de las clases debemos incluir clases que sean candidatas a padecer *God Class*, *Feature Envy*, ambos o ninguno de acuerdo a algún oráculo de referencia.

Como fuente de las clases a evaluar en la detección de *Design Smells* se eligió un proyecto Apache de código abierto que se encuentra disponible en un repositorio público. De esta manera las clases mostradas a los evaluadores humanos

pueden ser consultadas muy legiblemente en su contexto proporcionando el enlace al repositorio de código. El proyecto Apache Lucene es uno de los más frecuentemente usados en las publicaciones sobre detección de *Design Smells* según un estudio realizado por estos autores. En concreto se ha utilizado la versión 3.1.0. El código de este proyecto se puede consultar en `http://grepcode.com/snapshot/repo1.maven.org/maven2/org.apache.lucene/lucene-core/3.1.0`. Concretamente las clases elegidas para este experimento fueron:

- org.apache.lucene.search.TopDocs,
- org.apache.lucene.queryParser.TokenMgrError,
- org.apache.lucene.util.ReaderUtil,
- org.apache.lucene.analysis.CharArraySet y
- org.apache.lucene.index.FieldsWriter

Los detalles del diseño de la encuesta realizada para obtener respuestas de los evaluadores humanos se pueden consultar en `http://www.infor.uva.es/~yania/designsmells/#survey` donde se ha alojado un modelo de la encuesta. Con la intención de disponer de respuestas de evaluadores con diferentes perfiles y llegar al grupo de personas que podrían considerarse expertos en el tema de *Design Smells*, se recopilaron los e-mails de contacto de los autores de artículos sobre *Design Smells*. Se escribió directamente a dichos autores solicitando su participación en el estudio. Adicionalmente, los canales de difusión de la encuesta fueron diversos, desde el colegio de Ingenieros de Informática, linkedin y colegas de diferentes universidades; además se solicitó la difusión en un grupo de empresas de la región de Bélgica, Holanda y Alemania, a través de un colega con contactos profesionales en dichos países. Aunque desconocemos la ratio sujetos_contactados/sujetos_incluidos, consideramos que disponer de 92 respuestas válidas es un buen resultado, comparado con los resultados de trabajos similares (ver Sección 5). El tiempo en que la encuesta estuvo abierta para recibir respuestas fue de 3 meses.

4. Estudio y análisis de las preguntas de investigación

Los sujetos que respondieron a la encuesta online fueron 93, de los que seleccionamos 92, pues una de ellas fue no válida. Así pues, dispondremos de 92 sujetos como evaluadores humanos de las 5 clases seleccionadas, que las clasificarán con *Design Smell* (*God Class* o *Feature Envy*) o sin *Design Smell*.

El grado de acuerdo entre los diferentes evaluadores lo examinamos con el coeficiente *Kappa-Fleiss* que permite comparar el grado de concordancia de r evaluadores en k objetos evaluados. La interpretación de este coeficiente se muestra en la Tabla 1. Hemos utilizado la herramienta R con la que se obtiene este coeficiente de concordancia usando el paquete `irr`, que contiene la función `kappam.fleiss()`. Esta función además nos proporciona un test de hipótesis para aceptar (o rechazar) que no hay concordancia entre los evaluadores.

Se obtuvieron respuestas de diferentes países. Bélgica, Canadá, India, Estados Unidos, Holanda, República Checa, Israel, Pakistán, Corea del Sur, Líbano, Malasia, Nueva Zelanda, Marruecos, Reino Unido, Emiratos Árabes, Jordania, Arabia Saudí, Suiza y España. Como la dispersión de los 92 sujetos por los 20

Valor de kappa, grado de concordancia	
$k < 0{,}20$	Pobre
$0{,}21 \leq k < 0{,}40$	Débil
$0{,}41 \leq k < 0{,}60$	Moderada
$0{,}61 \leq k < 0{,}80$	Buena
$0{,}81 \leq k \leq 1{,}00$	Muy buena

Tabla 1: Interpretación de los valores del coeficiente *Kappa-Fleiss* utilizada en el estudio.

Europa	Asia	América	África	Oceanía
35	13	22	7	15

Tabla 2: Distribución de sujetos por continente.

países no deja suficientes sujetos/pais para focalizar el estudio por paises, decidimos analizar por áreas geográficas basadas en continentes. De esta manera se decidió diferenciar entre: Europa, Asia, América, África y Oceanía. La distribución del número de respuestas por continente se muestra en la Tabla 2. Se puede ver que en Europa es donde se localiza el mayor número de sujetos, y en África donde menos.

Por otra parte, la distribución de los sujetos por actividad laboral (Tabla 3) presenta la mayor frecuencia de sujetos en el Grupo 1 (Professor, Lecturer, Instructor) y la menor en el Grupo 4 (Software Architect). Cuando analizamos las respuestas obtenidas en el Grupo "Other", encontramos "System admin", "Quality Consultant", "Engineer" y algún que otro "Msc Student" o "Student". Esto nos motivó a dirigir los análisis posteriores clasificando esta variable en dos grupos: Industria y Academia, agrupando el Grupo 1 con el Grupo 2 como Academia, el grupo 3 y el grupo 4 como Industria y del Grupo 5, repartiendo a los estudiantes como Academia y al resto de respuestas en dicho Grupo 5 como Industria.

Mirando la distribución conjunta, el grupo con mayor número de sujetos fue: (Industria, entre 5 y 10 años de experiencia). El de menor frecuencia fue: (Academia, menos de 5 años de experiencia). Al considerar las distribuciones marginales, hubo más sujetos procedentes de la Industria que de la Academia. Además, el grupo de sujetos con entre 5 y 10 años de experiencia fue el más frecuente, y el de más de 10 años el que menos.

En la Tabla 5 se muestran tres distribuciones de frecuencias correspondientes a considerar el grado de formación condicionado por 3 factores determinados por los conocimientos sobre Escribir código Orientado a Objetos (OO), Revisar código OO y Experiencia en *Design Smells* (en la encuesta utilizamos el término *Code Smell* por ser más extendido entre los desarrolladores). Se puede ver que el mayor número de sujetos se concentra en aquellos cuyo grado de experiencia es Intermedio, sea cual sea el otro factor.

En las Tablas 6 y 7 se muestran los resultados obtenidos en la encuesta sobre el esfuerzo de los sujetos al detectar los *Design Smells*. Este esfuerzo se evalúa de dos formas. Por una parte en la Tabla 6 se presenta la distribución de frecuencias

Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5
35	13	22	7	15

Grupo 1 Professor, Lecturer, Instructor
Grupo 2 Researcher
Grupo 3 Programmer
Grupo 4 Software Architect
Grupo 5 Other

Tabla 3: Distribución de sujetos por actividad laboral.

Activities/Experience	1	2	3	Total	%
Academia	11	13	16	40	43,5 %
Industria	17	24	11	52	56,5 %

1 Menos de 5 años de experiencia
2 Más de 5 y menos de 10 años
3 Más de 10 años de experiencia

Tabla 4: Distribución conjunta de los sujetos por experiencia*actividad profesional.

del tiempo empleado en detectar los *Design Smells*. A los encuestados se les informó de que necesitarían una media de 25 minutos para realizar dicha tarea, y aproximadamente un 80 % tardaron menos de 25 minutos. La media de tiempo por clase fue de 6 minutos, lo cual confirma que detectar *Design Smell* sin ayuda de herramientas en proyectos grandes, es inviable. En cuanto a la Tabla 7, la mayoría de los sujetos (al menos un 79 %) consideran que tanto la revisión del código desarrollado por terceros, como la detección de los *Design Smells* tienen una dificultad de media a difícil.

Con el fin de contestar a las cuestiones de esta investigación, vamos a utilizar el coeficiente *Kappa-Fleiss*, y el test de hipótesis relativo a la no concordancia, con un nivel de significación de 0,05. La hipótesis nula se define como:

H_0 No existe acuerdo entre evaluadores ($kappa = 0$)

Esta hipótesis se subdivide de la siguiente forma:

H_0^{GC} No existe acuerdo entre evaluadores el detectar *God Class*

H_0^{FE} No existe acuerdo entre evaluadores el detectar *Feature Envy*

Estudiaremos primero los test para la Hipótesis 1a, relativa a la concordancia entre sujetos. Y después la Hipótesis 1b, relativa a la concordancia entre sujetos y herramientas. Y en cada caso, el estudio se hará para cada *Design Smell* considerado, *God Class* y *Feature Envy*: Para hacer este análisis se han realizado más de 100 estudios con el coeficiente *Kappa-Fleiss*, con diferentes combinaciones relativas al perfil de los evaluadores. Vamos a agrupar las preguntas de investigación en dos grupos. En el primer grupo, las que afectan a una comparación entre evaluadores humanos, y en el segundo grupo, las que afectan a una comparación entre los evaluadores humanos y las herramientas.

RQ1 ¿Existe acuerdo entre los evaluadores humanos al detectar *Design Smells*?

Se realiza un estudio del grado de acuerdo entre evaluadores (inter raters) con el estadístico *Kappa-Fleiss*. Los resultados de los p-valores y el coeficiente obtenido para cada caso (*God Class* y *Feature Envy*) se pueden ver en la Tabla 8. En el caso de *God Class* se obtiene un p-valor cercano al límite establecido para ser significativo (0,05) y el valor del coeficiente según la interpretación indica que el grado de acuerdo es Pobre. En el caso de *Feature Envy* el p-valor indica que no se puede rechazar la hipótesis nula por lo que se puede decir que no hay acuerdo al detectar, el grado de coincidencia es peor que una evaluación realizada al azar.

Experiencia/Con	Escribir Código OO	%	Revisar Código OO	%	*Design Smells*	%
None	9	9,8 %	8	8,7 %	19	20,7 %
Principiante	14	15,2 %	17	18,5 %	26	28,3 %
Intermedio	44	47,8 %	46	50 %	35	38 %
Avanzado	25	27,2 %	21	22,8 %	12	13 %

Tabla 5: Distribución de frecuencias de la experiencia con las actividades Escribir Código OO, Revisar Código OO y *Design Smells*.

Porcentaje de evaluadores por intervalos de tiempo en minutos								
0–4	5–9	10–14	15–19	20–24	25–29	30–34	35-39	40–45
9,8 %	19,6 %	21,7 %	17,4 %	10,9 %	1,1 %	4,3 %	0 %	3,3 %

Tabla 6: Distribución de frecuencias del tiempo empleado en detectar los *Design Smells*.

RQ5 ¿Cómo afecta el grado de experiencia en la detección de *Design Smells*?

RQ5a ¿El grado de acuerdo entre evaluadores humanos es mayor en el grupo de evaluadores con mayor experiencia?

Las primeras exploraciones sobre el acuerdo por grupos indicaron que la significación aumentaba si comparábamos el grupo de Inexperiencia (Menos de 5 años) con el grupo de Experiencia (Más de 5 años). Por esta razón, en lugar de establecer tres grupos de experiencia como se preguntaba en la encuesta, se dividieron los datos en dos grupos: menos de 5 años de experiencia y más de 5 años de experiencia. En la Tabla 9 se muestran los resultados del estudio del coeficiente *Kappa-Fleiss* y los p-valores obtenidos. Solamente se obtienen resultados significativos cuando el grupo de más de 5 años de experiencia detecta *God Class* pero el grado de acuerdo es muy pobre. En el resto de los casos no se puede rechazar la hipótesis nula por lo que no hay acuerdo entre los evaluadores.

RQ6 ¿Cómo afectan los antecedentes (en términos de formación, experiencia y conocimientos) y el contexto de los evaluadores en la detección de *Design Smells*?

RQ6a ¿Afecta el contexto de trabajo? ¿el área geográfica donde se forma o desarrolla la persona? ¿si se trata de un contexto académico o de la industria?

Los estudios del grado de acuerdo al detectar los *Design Smells God Class* y *Feature Envy*, teniendo en cuenta el área geográfica, indican que este no varía de un área a otro, excepto para Europa. De hecho, todos los resultados (excepto en Europa) son no significativos, obteniéndose p-valores bastante grandes en todos los casos (por cada área, por cada *Design Smell*). El único caso en el que se obtiene un resultado significativo es en Europa al detectar *God Class*. El p-valor es próximo a cero y el valor de *Kappa-Fleiss* es de 0,18 muy próximo a lo que se interpreta como acuerdo "débil".

Porcentaje de evaluadores por medición de la dificultad en la tarea						
Tarea/Dificultad	No contesta	Muy fácil	Fácil	Medio	Difícil	Muy difícil
Revisar el código	4,3 %	6,5 %	14,1 %	51,1 %	19,6 %	4,3 %
Detectar *Design Smells*	4,3 %	7,6 %	7,6 %	50 %	22,8 %	7,6 %

Tabla 7: Distribuciones de frecuencias de la dificultad encontrada al Revisar el código y al Detectar *Design Smells*.

H_0	p-value	*Kappa-Fleiss*	Conclusión
H_0^{GC}	0,0565	−0,00536	Cercano a significativo pero el grado de acuerdo al detectar *God Class* es muy pobre
H_0^{FE}	0,438	0,0132	No significativo. No hay acuerdo al detectar *Feature Envy*

Tabla 8: Resultados del estudio del grado de acuerdo entre evaluadores al detectar *God Class* y *Feature Envy*

Los test realizados para estudiar el efecto del entorno (Académico vs. Industria) en el grado de acuerdo al detectar los *Design Smells God Class* y *Feature Envy*, fueron todos no significativos, no había acuerdo. Por lo tanto no parece que el entorno influya en el grado de acuerdo entre sujetos al detectar *God Class* y *Feature Envy*.

RQ6b ¿Afectan los antecedentes del evaluador? Los antecedentes del evaluador se miden respecto a su experiencia en programación orientada a objetos, en lecturas o revisiones de código, y respecto a su nivel de conocimiento de *Design Smells*

En la Tabla 10, los casos resaltados en gris con el p-valor $\geq 0{,}05$ indican que no se puede rechazar la hipótesis nula por lo tanto se asume que no hay acuerdo entre los evaluadores. Sin embargo en los otros casos, se puede admitir que hay acuerdo, que se puede interpretar mirando el valor del coeficiente *Kappa-Fleiss* como acuerdo muy pobre. Como se puede ver el acuerdo se produce en el grupo de mayor experiencia cuando detecta *God Class* y, sin embargo, en el grupo de los más inexpertos cuando detectan *Feature Envy*.

Es curioso que se repite el mismo patrón para los tres tipos de actividad. Una posible explicación puede ser que en el *Design Smell God Class* se relaciona con los conceptos de tamaño y de complexidad ciclomática. Sin embargo, *Feature Envy* se relaciona con conceptos de cohesión y acoplamiento en código OO, con la violación de los patrones GRASP. Suponemos que los más jóvenes están formados en estos conceptos en sus cursos de Ingeniería del Software y Programación. Ellos están al tanto de estos problemas, de ahí que tengan algún acuerdo detectando *Feature Envy*, aunque pobre. Sin embargo, los más experimentados tienen mejor desarrollados los aspectos relativos al manejo del tamaño y la complejidad. Los principiantes tienden a producir métodos y clases más largas y complejas. La inexperiencia les lleva a creer que pueden manejar el tamaño que los expertos consideran demasiado grande.

RQ2 ¿Existe acuerdo entre los evaluadores humanos y las herramientas de detección de *Design Smells*?

H_0	p-value	*Kappa-Fleiss*	Conclusión
H_0^{GC}	Inex. 0,215 Exp. 0,00204	Inex. −0,028 Exp. 0,0307	Significativo cuando los más expertos detectan *God Class* pero el acuerdo es muy pobre
H_0^{FE}	Inex. 0,524 Exp. 0,267	Inex. −0,0146 Exp. −0,0111	No significativo. No hay acuerdo al detectar *Feature Envy* en ningún caso

Tabla 9: Resultados del grado de acuerdo entre evaluadores del mismo grupo de menos de 5 años vs más de 5 años de experiencia

	Inexpertos (None+Beginner)			con Experiencia (Intermediate+Expert)		
Exp. Escribir código OO	DS	*p*	*k*	DS	*p*	*k*
	God Class	0,579	0,015	*God Class*	0	0,117
	Feature Envy	0,008	0,078	*Feature Envy*	0,605	0,005
Exp. Revisar código OO	DS	*p*	*k*	DS	*p*	*k*
	God Class	0,159	0,036	*God Class*	0,011	0,024
	Feature Envy	0,005	0,073	*Feature Envy*	0,399	0,008
Exp. con *Design Smells*	DS	*p*	*k*	DS	*p*	*k*
	God Class	0,960	0,001	*God Class*	0,006	0,001
	Feature Envy	0,527	0,009	*Feature Envy*	0,101	0,022

Tabla 10: Resultados del estudio por grupos de sujetos, clasificados por experiencia, respecto de diferentes actividades (Escribir código OO, Revisar código OO y Detección de *Design Smells*)

Para contestar a esta pregunta, se compara el resultado de las personas encuestadas con las mismas 6 herramientas utilizadas en el primer experiemento que se presenta en [1] comparando las herramientas de detección, en este caso: Together, JDeodorant, iPlasma, inFusion, inCode y JSensorSmell cuando detectan *God Class* y *Feature Envy* en las clases del proyecto Apache Lucene, en particular sus resultados sobre las 5 clases que fueron preguntadas en la encuesta.

Los tests realizados indican que no existe acuerdo entre evaluadores humanos y herramientas en ninguno de los dos casos (*God Class*, *Feature Envy*). Los p-valores que se obtienen son 0,111 y 0,51, respectivamente.

RQ4 ¿Qué herramienta está más cerca de los evaluadores humanos al detectar *Design Smells*?

Cuando se estudia el grado de acuerdo entre los sujetos y cada herramienta por separado, para detectar *God Class* y *Feature Envy*, se obtuvieron en todos los casos resultados no significativos, esto es, no hubo acuerdo. Sin embargo, en el caso del *God Class*, los p-valores estaban próximos a ser significativos (cercanos a $0,05$) en las 6 herramientas. Pero aún en estos casos, los valores del coeficiente *Kappa-Fleiss* darían un acuerdo muy pobre, oscilan entre 0,0617 y 0,0776. Se concluye que ninguna de las herramientas está más cerca de los evaluadores humanos en general, pero todas se acercan más en el caso de *God Class* frente a *Feature Envy*.

RQ5b ¿El grado de acuerdo con las herramientas de detección es mayor cuando el evaluador humano tiene mayor experiencia?

Se ha realizado un estudio cruzado, de cada herramienta con los diferentes grupos de evaluadores basados en grado de experiencia en las actividades relevantes para la detección de *Design Smells*, área geográfica de procedencia, contexto de trabajo Academia o Industria, años de experiencia en sus actividades profesionales. Por razones de espacio no podemos exponer todos los datos obtenidos pero podemos resaltar que en cada herramienta se obtuvo un patrón similar, que describiremos a continuación:

Se obtienen resultados significativos que permiten rechazar la hipótesis nula aunque los coeficientes *Kappa-Fleiss* sean muy pequeños y por tanto el acuerdo pobre para un perfil de evaluador que se describe como sigue:

- Evaluadores expertos en la actividad de escribir código OO y/o en revisión de código OO cuando detectan *God Class*.
- Evaluadores principiantes en la actividad de escribir código OO y/o en revisión de código OO cuando detectan *Feature Envy*.
- Evaluadores expertos en *Design Smells* cuando detectan *God Class*.
- Evaluadores de Europa cuando detectan *God Class*.

5. Trabajos relacionados

No se cuenta con muchos estudios empíricos sobre detección de *Design Smells* por evaluadores humanos o que comparen los resultados de evaluadores humanos con la detección realizada por herramientas.

Mäntylä en [7] describe dos experimentos realizados entre los años 2003 y 2004, publicados posteriormente en un artículo que aparece en el año 2005. En el primero de ellos, realizó una encuesta preguntando a un grupo de evaluadores la presencia en el código de tres *Design Smells*. Se trataba de tres *Design Smells* de nivel de método: *Long Method*, *Long Parameter List* y *Feature Envy*. Se preguntaba además a los evaluadores si pensaban que el método debía ser refactorizado para eliminar los *Design Smells*. En el segundo experimento, no se preguntó por la presencia de los *Design Smells* sino directamente si el método debía ser refactorizado. En el primer experimento participaron 46 evaluadores y en el segundo 36. Los evaluadores eran estudiantes de Máster de los cuales poco más del 50 % tenían cierta experiencia desarrollando en la industria. El resultado del primer experimento reveló altos niveles de acuerdo en los evaluadores en los dos primeros *Design Smells* más simples y considerablemente débiles al detectar *Feature Envy* y al decidir si había que refactorizar el método. En el segundo experimento se obtuvo un acuerdo débil en la única pregunta sobre si había que refactorizar el método.

Mäntylä et al [6] en el año 2004 realizó un experimento en una pequeña empresa de desarrollo de software en Finlandia. De los 18 desarrolladores contestaron a su encuesta 12. En la encuesta preguntó a las desarrolladores por la presencia de 23 *Design Smells* en el código de varios módulos desarrollados en la propia empresa. Cada evaluador aportó la evaluación de una media de 3 módulos, cada módulo obtuvo una media de cerca de 4 evaluaciones. A pesar

de los pocos datos concluyeron que los desarrolladores líderes detectaban más *Design Smells* de tipo estructural mientras que los desarrolladores regulares detectaban más smells del tipo *Duplicate Code* o *Dead Code*. Encontraron mucha subjetividad en la evaluación. Para tres *Design Smells* como *Large Class*, *Long Parameter List* y *Duplicate Code* realizaron un estudio con métricas de código y concluyeron que la evaluación subjetiva de los desarrolladores no correlacionaba con las métricas de código.

En 2012 Yamashita y Moonen [13] realizaron un estudio teniendo en cuenta las opiniones de 2 expertos externos a una organización y de 6 desarrolladores obtenidos de un proceso de selección para encargarles el mantenimiento de dos sistemas. Con esto obtuvieron la opinión de evaluadores humanos sobre la mantenibilidad del sistema antes y durante su mantenimiento. Es un artículo muy interesante de leer. Se identificaron 13 factores importantes de cara a la mantenibilidad, en 9 de ellos coincidieron los expertos y los desarrolladores encargados del mantenimiento. Esta información la intentaron correlacionar con la detección de *Design Smells* como indicador de mantenibilidad. Obtuvieron correlaciones parciales con algunos *Design Smells*, analizaron de esos cuáles pueden ser detectados automáticamente por herramientas como: *God Class*, *God Method*, *Lazy Class*, *Message Chain*, *Long Parameter List*, *Duplicate Code*, *Switch statements*, *Feature Envy*, *Shotgun Surgery*, *ISP Violation* para dar una visión cuantitativa de aspectos cualitativos como la mantenibilidad.

Nuevamente Yamashita y Moonen en 2013 [14] diseñaron una encuesta para explorar si los *Design Smells* son importantes para los desarrolladores y en caso de que no, si se debe a irrelevancia del concepto, desconocimiento por parte de los desarrolladores o carencia de herramientas apropiadas para detectar *Design Smells* y eliminarlos. Encuestaron a 85 desarrolladores profesionales contratándolos para realizar la encuesta a través de un portal de mercado de trabajo para freelance. El 32 % contestó que nunca habían oído hablar de *Design Smells* o término similar. El 22 % contestó que lo había oído o visto en algún blog pero no estaba seguro de saber lo que era. Esto resulta en más del 50 % con prácticamente ningún conocimiento sobre el tema. El 21 % conoce el concepto pero no lo aplica. Solamente el 18 % declara tener buenos o sólidos conocimientos sobre el tema y aplicarlo en sus actividades diarias. También elaboraron un ranking de los *Design Smells* más conocidos por los encuestados. Sólo 2 de los 85 encuestados respondieron que usaban alguna herramienta de detección de *Design Smells*. Concluyeron los autores de forma muy similar a nuestro estudio, se necesita formación y divulgación, y se necesitan herramientas con algunas características deseables entre las que destacamos estar lista para usar pero con reglas configurables.

Palomba et al [8] publicaron en 2014 un estudio con 34 sujetos entre Estudiantes de Máster (15), Desarrolladores de Proyectos de Código abierto (10) y Desarrolladores trabajando en la industria del software (9). El objetivo del estudio era saber hasta qué punto los desarrolladores perciben los *Design Smells* como problemas de diseño que hay que solucionar y cuáles de estos *Design Smells* eran considerados más dañinos. Algunos *Design Smells* no fueron percibidos como problemas de diseño a solucionar. Los *Design Smells* relacionados con la complejidad y el tamaño como *God Class* son percibidos siempre como un problema, y en particular *God Class* fue el ganador en el ranking de los considerados

como más dañinos. El caso de *Feature Envy* es también muy interesante, es uno de los que más variabilidad presenta en las respuestas de los encuestados. Los autores concluyen en ese caso que se trata de posibles abusos/malos usos de los principios del diseño Orientado a Objetos.

6. Conclusiones y trabajo futuro

En este documento hemos presentado los resultados obtenidos al realizar una encuesta online, con el fin de que sujetos de diferentes perfiles y procedencias, identificaran los *Design Smells God Class* y *Feature Envy*, en 5 clases JAVA. A partir de esta información, hemos estudiado el grado de acuerdo entre dichos sujetos al detectar los *Design Smells* mencionados. Además, hemos estudiado el grado de acuerdo al detectar *Design Smells* entre sujetos y 6 herramientas, de las que se habían obtenido los datos en un estudio previo. La principal conclusión a la que llegamos es que hemos confirmado nuestra sospecha de que no existe acuerdo entre los desarrolladores en general y entre los desarrolladores y las herramientas, como consecuencia de lo anterior. En los casos en los que se produce algo de acuerdo, éste es débil o pobre, muy pobre en la mayoría de los casos.

Se detecta que existe un perfil de evaluador en el que se dan los mejores casos de acuerdo entre sí y con las herramientas. Este perfil nos indica que los desarrolladores experimentados coinciden mejor en cuestiones relativas a tamaño y complejidad, que los menos experimentados tienen más reciente el conocimiento sobre principios y patrones del diseño Orientado a Objetos y que se necesita mayor formación en *Design Smells*. En nuestro estudio el 49 % de los encuestados se declara sin experiencia o principiante en cuanto a *Design Smells*. Solamente el 13 % indica que se considera experto en la materia y esto considerando que hemos hecho un esfuerzo por alcanzar expertos en la materia mediante la elaboración de una lista de correo con los autores de artículos sobre detección de *Design Smells*.

Los resultados observados nos permiten elaborar algunas recomendaciones, para propiciar la adopción de técnicas de detección de *Design Smells* en la industria:

- Se debe incorporar formación sobre *Design Smells* en los estudios que preparan a los futuros desarrolladores de software e investigadores en Ingeniería del Software,
- Se debe de contar con benchmarks consensuados, con el fin de asegurar que las herramientas cumplen unos mínimos que garanticen su utilidad para detectar *Design Smells*.
- Se necesita contar con la opinión de profesionales expertos en aquellas actividades relacionadas con la detección de *Design Smells*, como escribir código, revisar o leer código desarrollado por otros, y en conocimiento de *Design Smells* a la hora de validar los resultados de las herramientas de detección.

Como trabajo futuro, pensamos que es de interés realizar una réplica, con sujetos de perfil más homogéneo, y alta experiencia en los aspectos que son relevantes en la detección de *Design Smells*.

Referencias

1. AlKharabsheh, K., Crespo, Y., Manso, E., Taboada, J.: Comparación de herramientas de detección de design smells. In: JISBD'2016, Salamanca, Septiembre (2016)
2. Brown, W.J., Malveau, R.C., McCormick III, H.W., Mowbray, T.J.: AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. John Wiley and Sons (March 1998), http://www.antipatterns.com
3. Fleiss, J.L.: Statistical methods for rates and proportions. New York: John Wiley, 2nd edn. (1981)
4. Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: Refactoring: Improving the Design of Existing Code. Object Technology Series, Addison-Wesley (June 1999)
5. López, C., Manso, E., Crespo, Y.: Evaluación de la eficiencia de métodos de identificación del defecto de diseño God Class. In: XVII Jornadas de Ingeniería del Software y de Bases de Datos (JISBD 2012). Universidad de Almería (2012), http://www.giro.infor.uva.es/Publications/2012/LMC12
6. Mäntylä, M., Lassenius, C.: Subjective evaluation of software evolvability using code smells: An empirical study. Empirical Software Engineering 11(3), 395–431 (2006)
7. Mäntylä, M., Vanhanen, J., Lassenius, C.: Bad smells - humans as code critics. In: 20th International Conference on Software Maintenance (ICSM 2004), 11-17 September 2004, Chicago, IL, USA. pp. 399–408 (2004)
8. Palomba, F., Bavota, G., Penta, M.D., Oliveto, R., Lucia, A.D.: Do they really smell bad? A study on developers' perception of bad code smells. In: 30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014. pp. 101–110 (2014)
9. Riel, A.J.: Object-Oriented Design Heuristics. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (April 1996)
10. Tiberghien, A., Moha, N., Mens, T., Mens, K.: Répertoire des défauts de conception. Tech. Rep. 1303, University of Montreal (2007)
11. Wake, W.C.: Refactoring Workbook. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2003)
12. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A.: Experimentation in Software Engineering: An Introduction, International Series in Software Engineering, vol. 6. Springer (2000)
13. Yamashita, A.F., Moonen, L.: Do code smells reflect important maintainability aspects? In: 28th IEEE International Conference on Software Maintenance, ICSM 2012, Trento, Italy, September 23-28, 2012. pp. 306–315 (2012)
14. Yamashita, A.F., Moonen, L.: Do developers care about code smells? an exploratory survey. In: 20th Working Conference on Reverse Engineering, WCRE 2013, Koblenz, Germany, October 14-17, 2013. pp. 242–251 (2013)

Comparación de herramientas de Detección de *Design Smells*

Khalid AlKharabsheh*[1], Yania Crespo[2], Esperanza Manso[2], and José Ángel Taboada[1]

[1]{khalid,joseangel.taboada}@usc.es, CiTIUS
Centro Singular de Investigación en Tecnologías de la Información
Universidad de Santiago de Compostela, Santiago de Compostela 15782
[2]{yania,manso}@infor.uva.es, Dpto. Informática, Universidad de Valladolid
Escuela de Ingeniería Informática. Campus Miguel Delibes, Valladolid 47011

Resumen La detección de *Design Smells* ha experimentado un auge en actividad entre los años 2010 y 2014. Proliferan las herramientas de detección automática pero se percibe un problema de falta de acuerdo en la identificación de *Design Smells*. En este trabajo se presentan dos experimentos. El primero es un experimento diseñado como estudio preliminar en el que se comparan una selección de 6 herramientas de detección de *Design Smells*. En este primer experimento el estudio se realizó centrándose en la detección de dos tipos de *Design Smells*: *God Class* y *Feature Envy* en un proyecto software de código abierto. Del proyecto seleccionado se eligieron 100 clases aleatoriamente para este primer estudio exploratorio. El análisis consistió en valorar el grado de acuerdo en la identificación de *Design Smells* en el grupo de herramientas. Para profundizar en el problema se diseñó una réplica del primer experimento. En esta réplica se comparan 5 herramientas de detección de *Design Smells*. En este segundo experimento se analizaron 12587 clases fruto de la preparación de un dataset con todas las clases de 24 proyectos de código abierto obtenidos de SourceForge. Este segundo experimento se centró únicamente en el estudio del acuerdo entre las herramientas en la identificación de *God Class*. Los resultados obtenidos en este segundo experimento muestran que tanto el acuerdo entre las herramientas tomadas conjuntamente como analizadas dos a dos es pobre o débil según el caso, de acuerdo a la escala de interpretación de los valores del estadístico *Kappa-Fleiss*.

Keywords: Detección de *Design Smells*, *Kappa-Fleiss*, FCA, calidad, evolución, mantenimiento

1. Introducción

Un *Design Smell* es un problema encontrado en la estructura del software (código, diseño) que no produce errores de compilación o de ejecución, pero afecta negativamente los factores de calidad del software. *Design Smell* es un término unificador que agrupa en un único concepto a *Code Smells*, *Bad Smells*,

* Khalid AlKharabsheh agradece la financiación del programa Erasmus Mundus

Disharmonies, *Antipatterns*, *Design flaws*, *Technical debt*, entre otros términos conocidos en el estado del arte actualmente.

La detección de *Design Smells* tienes sus inicios a partir del año 2000 y ha experimentado un incremento notable de actividad entre los años 2010 y 2015. En el año 2011 como resultado de un proyecto realizado en nuestro grupo de investigación se publicaron dos informes técnicos [12,13] en los que se analizaron y clasificaron las herramientas propuestas hasta el momento para la detección y corrección de *Design Smells*. Una vez clasificadas dichas herramientas, se descartan aquellas que se constata detectan únicamente problemas de estilo al escribir código así como aquellas que solamente realizan labores de corrección, basadas en *refactoring*. De esta forma se obtiene que las herramientas disponibles hasta el 2010 para la detección de *Design Smells* son: Analyst4j, Cultivate (jTransformer suite), DÉCOR, iPlasma, inCode, inFusion, jDeodorant, una incipiente PMD, Reek, RevJava, SA4J y Together, fundamentalmente. También en el año 2011, Zhang et al [19] publicaron una revisión sistemática del trabajo realizado en cuanto a *Code Smells* desde el año 2000 hasta el 2009 , analizando en detalle 39 trabajos relevantes en la materia pero no se detiene en la revisión de las herramientas disponibles hasta el momento.

Entre el 2010 y el 2015, nuevos estudios se realizan para organizar el conocimiento producido en esta materia. En [14] se aborda una revisión de las técnicas de detección de *Code Smells* publicadas a lo largo de los 15 años de trabajo en la materia desde el 2000 hasta los inicios del 2015. En este estudio puede consultarse la aparición de un gran grupo de herramientas como: Stench Blossom, ConcernReCS, SourceMiner, BSDT, JCodeCanine, GrouMiner, CodeVizard, JSNose, Hist-Inspect, SVMDetect, PTIDEJ suite (que contiene DÉCOR y su evolución DETEX), BLOP, una evolución de la anteriormente incipiente PMD con reglas extendidas para este particular, así como un gran grupo de prototipos de investigación sin un nombre particular que implementan las técnicas trabajadas por sus autores.

Nos encontramos con todo tipo de herramientas para detectar *Design Smells*: herramientas dedicadas, herramientas integradas como plug-ins en entornos de desarrollo, aplicando técnicas diferentes, para varios lenguajes de programación. Sin embargo, a pesar de este auge, no se ha experimentado una adopción de estas herramientas por parte de la industria comparable con la que sí se constata de las herramientas de *refactoring*.

Teniendo en cuenta nuestros conocimientos previos y la revisión del estado del arte, nuestro grupo exploró la cuestión de modelar la subjetividad en la detección de *Design Smells*, introduciendo información semántica sobre la intención de diseño del desarrollador, así como teniendo en cuenta históricos de datos de las organizaciones y la retroalimentación de falsos negativos y falsos positivos detectados por expertos [11]. A partir de todos estos antecedentes, han surgido nuevas líneas de investigación en relación con el uso de herramientas en la detección de *Design Smells*, que hemos abordado en este documento.

En este trabajo se presentan dos experimentos. El primero es un experimento diseñado como estudio preliminar en el que se comparan una selección de 6 herramientas (inCode, inFusion, iPlasma, Together, JDeodorant y JSmellSensor) usándolas para detectar *Design Smells* en un proyecto software de código abierto. Del proyecto seleccionado se eligieron 100 clases aleatoriamente para este

primer estudio exploratorio. El análisis consiste en valorar el grado de acuerdo entre el grupo de herramientas en la identificación de *Design Smells*. En este primer experimento el estudio se realizó centrándose en la detección de dos tipos de *Design Smells*: *God Class* y *Feature Envy*. De este primer estudio se detectó un grupo de tres herramientas (inCode, inFusion, iPlasma) con un grado de acuerdo muy bueno. Este hecho puede explicarse debido a que las tres fueron desarrolladas en el mismo grupo de investigación. Entre el resto de herramientas o bien el acuerdo es inexistente (peor que al azar) o bien es débil. El acuerdo débil se detectó entre el grupo de las tres herramientas antes mencionadas y la herramienta JSmellSensor. JSmellSensor es una herramienta experimental desarrollada en nuestro grupo de investigación que parte de conocimiento aportado por inCode. Debido a estas circunstancias y para profundizar en el problema se diseñó una réplica del primer experimento. En esta réplica se comparan 5 herramientas (iPlasma, Together, JDeodorant, PMD, DÉCOR). Del grupo de tres herramientas identificado en el primer experimento se eligió como representante iPlasma. Se decidió descartar JSmellSensor para esta réplica dado su carácter experimental e interno a nuestro grupo. En este segundo experimento se analizaron 12587 clases fruto de la preparación de un dataset con todas las clases de 24 proyectos de código abierto obtenidos de SourceForge. Este segundo experimento se centró únicamente en el estudio del acuerdo en la identificación de *God Class*. Los resultados obtenidos en este segundo experimento muestran que tanto el acuerdo entre las herramientas tomadas conjuntamente como analizadas dos a dos es pobre o débil según el caso, de acuerdo a la escala de interpretación de los valores del estadístico *Kappa-Fleiss* [3] que se utiliza en los análisis de ambos experimentos.

El resto del artículo se organiza de la siguiente forma para mostrar los detalles del trabajo realizado. En la Sección 2 se describe el marco conceptual para este trabajo, tratando cuestiones relativas a la detección de los defectos de diseño que nos ocupan. En la Sección 3 se detallan los objetivos del trabajo y la planificación del primer experimento. La Sección 4 presenta los estudios realizados sobre los resultados de las herramientas de detección de *Design Smells* seleccionadas y el análisis de los mismos. A partir de estos resultados se planifica una réplica del anterior con los mismos objetivos y preguntas de investigación que se describe en la Sección 5 En la Sección 6 se describen brevemente algunos trabajos relacionados mientras que la Sección 7 presenta un resumen de las conclusiones obtenidas y las líneas de trabajo a partir del mismo.

2. Detección de los *Design Smells God Class* y *Feature Envy*

El estudio se ha realizado centrándose en la detección de *God Class* (*GC*) y *Feature Envy* (*FE*). Se trata de dos tipos diferentes de *Design Smells*. *God Class* es un *Design Smell* intra clase, sólo se necesita observar el código de dicha clase para detectarlo. Por su parte *Feature Envy* es un *Design Smell* inter clase, para detectarlo es relevante observar la interacción de dicha clase con otras clases relacionadas.

En los buenos diseños orientados a objetos la lógica del sistema está distribuida uniformemente entre las clases de los niveles superiores. En [15] se define una

God Class como un objeto que controla demasiados objetos en el sistema y que ha crecido más allá de toda lógica para convertirse en la clase que lo hace todo. En una *God Class* hay demasiadas variables de instancia y demasiado código. Donde hay demasiado código hay peligro de que surja código duplicado. A veces se corresponde con una mala aplicación del patrón de diseño Mediador. Este problema de diseño puede ser parcialmente asimilado con el defecto *Large Class* definido por Fowler [6]. También es conocido como el antipatrón *"The Blob"* en [1].

Feature Envy se recoge en el catálogo de Fowler [6] y se clasifica por Wake [17] en la categoría que denomina "entre clases" y la subcategoría "de responsabilidad". Un método tiene el defecto *Feature Envy* si parece estar más preocupado en manipular datos de otras clases que de la suya propia. Está relacionado con la responsabilidad de la clase, ya que contiene métodos que deberían estar en otra clase. Como excepciones Fowler indica que existen varios patrones de diseño que rompen esta regla, como el Visitante y el Estrategia.

Se trata entonces de dos *Design Smells* de ámbito diferente. *God Class* es de ámbito de clase, para detectarlo hay que revisar la clase como un todo, mientras que *Feature Envy* es de ámbito de método, por ello para detectarlo basta con revisar el interior de un método. De acuerdo con Tiberghien et al. [16] en su clasificación de *Design Smells*, *God Class* pertenece al grupo de los *"Bloaters"* (hinchadores) mientras que *Feature Envy* pertenece al grupo de los *"Couplers"* (acopladores).

3. Objetivos y preguntas de investigación

El problema objeto de este estudio parte de una hipótesis general basada en nuestra experiencia en *Design Smells* y en la revisión del estado del arte. A pesar del auge en la investigación en detección de *Design Smells*, no se ha producido una contundente adopción en la industria, tal y como ha ocurrido con las herramientas de *refactoring*. Este hecho es llamativo porque la detección de *Design Smells* está, en su origen, estrechamente relacionada con las operaciones de *refactoring*. La detección de *Design Smells* se definía como detección de oportunidades de *refactoring*. Las herramientas de *refactoring* han sido adoptadas en la industria como mecanismo automatizado de edición de código de alto nivel, y por las prácticas ágiles y el desarrollo dirigido por tests (TDD), como parte del ciclo *Red-Green-Refactor*.

La sospecha que nos hemos planteado es que la ausencia de adopción de las herramientas de detección de *Design Smells*, tiene como base la falta de acuerdo entre ellas y la subjetividad, que parece ser intrínseca al problema.

Utilizando el patrón **GQM**, ampliamente extendido en la investigación en ingeniería del software para definir los objetivos [18], el objetivo de este estudio se define como:

Analizar una colección de herramientas
Con el propósito de evaluarlas
Con respecto al grado de acuerdo para detectar los *Design Smells God Class* y *Feature Envy*
Desde el punto de vista de los investigadores

Métrica	Valor
NOP Number of Packages (Número de Paquetes)	15
NOC Number of Classes (Número de Clases)	533
NOM Number of Methods (Número de Métodos)	3997
LOC Lines of Code (Líneas de Código)	45416

Tabla 1: Caracterización del proyecto Apache Lucene en su versión 3.1.0

En el contexto del software alojado en repositorios de software libre

De este objetivo general derivamos la siguiente hipótesis de trabajo, y cuestiones a contestar.

Hipótesis 1 *No existe acuerdo entre las herramientas cuando detectan los Design Smells God Class y Feature Envy.*

RQ1 ¿Existe acuerdo entre las herramientas seleccionadas al detectar *Design Smells God Class* y *Feature Envy*?
RQ2 ¿Cuáles de las herramientas seleccionadas coinciden más al detectar *Design Smells God Class* y *Feature Envy*?

3.1. Diseño de los experimentos

El conjunto de clases que tienen que evaluar las herramientas seleccionadas para detectar los *Design Smells*, proceden de un proyecto Apache de código abierto que se encuentra disponible en un repositorio público. El proyecto Apache Lucene es uno de los usados con más frecuencia en las publicaciones sobre detección de *Design Smells*, según un estudio realizado por estos autores. En concreto se ha utilizado la versión 3.1.0. Esta versión se puede caracterizar como se muestra en la Tabla 1. El código fuente de dicho proyecto en la versión utilizada se puede obtener en `http://grepcode.com/snapshot/repo1.maven.org/maven2/org.apache.lucene/lucene-core/3.1.0`.

En primer lugar se realizó un cuasi-experimento de carácter exploratorio para posteriormente abordar un experimento como réplica de éste teniendo en cuenta las lecciones aprendidas en el primero.

El diseño de este primer cuasi-experimento ha utilizado como punto de partida un informe técnico [12] del grupo GIRO de la Universidad de Valladolid. En éste se evalúan un amplio conjunto de herramientas de detección y corrección de *Design Smells*. A partir de este informe, se prescinde de la herramientas que realizan corrección de *Design Smells* pero no detección, quedando 24 herramientas. Del estudio de sus características, seleccionamos aproximadamente el 50 %, aquellas que permiten detectar DS en código JAVA. Posteriormente filtramos aquellas que tienen en común un grupo de *Design Smell* y que pueden procesar un proyecto desarrollado con la misma versión de JAVA.

En la selección de herramientas nos hemos encontrado con nueva información, que añade nuevas sospechas a la falta de adopción de herramientas por la industria: no existe un corpus de *Design Smells* común a varias herramientas, como ocurre en el caso de las herramientas de *refactoring*. Hemos elegido *God*

Tool/Smell	*God Class*	*Feature Envy*	Tiene ambos
inFusion v.1.8.4	6	3	1
inCode v.2.0.7	6	3	1
JDeodorant v.5.0.13	79	31	4
Together v.12.6.0	14	6	4
iPlasma v.6.1	12	16	2
JSmellSensor	13	8	1

Tabla 2: Número de clases detectadas con *Design Smells* en todo el proyecto Apache Lucene 3.1.0 por cada una de las 6 herramientas seleccionadas en el primer quasi-experimento.

Class y *Feature Envy* para el estudio porque son los *Design Smells* más citados en la bibliografía según un estudio realizado por estos autores. Estos *Design Smells* además son detectados por varias herramientas con lo que se permite comparar un grupo de herramientas seleccionadas. En algunos contextos está presente *God Class* junto con *Large Class* pero hemos optado por *God Class* que es más común. La elección de *God Class* y *Feature Envy* para el estudio es interesante además ya que se trata de dos tipos de *Design Smell* diferentes desde diferentes perspectivas como se explicó en la Sección 2.

Siguiendo este proceso de selección, en este primer quasi-experimento exploratorio, se seleccionaron 5 herramientas de otros autores (inFusion, inCode, iPlasma, JDeodorant y Together) unida a una desarrollada en nuestro equipo (JSmellSensor) como parte de una Tesis Doctoral [9,10].

Los detalles de la planificación de la réplica se explicarán más adelante después de analizar los resultados del primer cuasi-experimento.

4. Estudio y análisis de las preguntas de investigación

RQ1 Existe acuerdo entre las herramientas seleccionadas al detectar *Design Smells God Class* y *Feature Envy*?

Se han utilizado 6 herramientas, que han examinado 563 clases del proyecto seleccionado, con el fin de detectar los *Design Smells God Class* y *Feature Envy*.

En la Tabla 2 se muestra el resumen de las clases en las que se ha detectado alguno de los *Design Smells God Class* y *Feature Envy* en todo el código del proyecto Apache Lucene 3.1.0. A primera vista dichas cifras nos dicen que no hay mucho acuerdo entre las diferentes herramientas. Se observa una excepción entre inFusion e inCode, que resulta que son dos herramientas diferentes desarrolladas por la misma institución intooitus (`https://www.intooitus.com/`).

A la vista del resumen en cifras que se muestra en la Tabla 2 se podría haber optado por excluir del estudio bien a inFusion, bien a inCode, dado que en general están dando el mismo número global de clases infectadas pero se decidió continuar adelante con las 6 herramientas.

El grado de acuerdo entre los diferentes evaluadores (herramientas o sujetos) lo examinamos con el coeficiente *Kappa-Fleiss*. De acuerdo con [2] en la Tabla 3 se muestra cómo interpretar, en el campo de la ingeniería del software,

Valor de kappa, grado de concordancia	
$k < 0{,}20$	Pobre
$0{,}21 \leq k < 0{,}40$	Débil
$0{,}41 \leq k < 0{,}60$	Moderada
$0{,}61 \leq k < 0{,}80$	Buena
$0{,}81 \leq k \leq 1{,}00$	Muy buena

Tabla 3: Interpretación de los valores del coeficiente *Kappa-Fleiss* utilizada en el estudio.

Design Smell	p-value	*Kappa-Fleiss*	Interpretación
Feature Envy (FE)	0.00601	0.0709	Acuerdo pobre
God Class (GC)	2.43e-13	0.189	Acuerdo pobre

Tabla 4: Resultados del test Kappa-Fleiss, al comparar el grado de concordancia de las 6 herramientas seleccionadas en el primer quasi-experimento cuando analizan el proyecto Apache Lucene 3.1.0.

el grado de concordancia entre evaluadores dado un valor del coeficiente *Kappa-Fleiss*. Hemos utilizado la herramienta R con la que se obtiene este coeficiente de concordancia usando el paquete `irr`, que contiene la función `kappam.fleiss()`.

La hipótesis nula H_0 se estudia en dos casos H_0^{GC} y H_0^{FE} según se trate de analizar el grado de acuerdo al detectar *God Class* (GC) o *Feature Envy* (FE).

H_0^{GC}: Las herramientas no coinciden al identificar *God Class* en las clases del proyecto Apache Lucene 3.1.0.

H_0^{FE} Las herramientas no coinciden al identificar *Feature Envy* en las clases del proyecto Apache Lucene 3.1.0.

En todos los casos la hipótesis nula se asumirá como "No hay concordancia entre las herramientas". Se establece un p-valor $\leq 0{,}05$ para rechazar la hipótesis nula. Es decir, si se obtiene un p-valor menor o igual a 0,05 el test es significativo y se rechaza la hipótesis nula. Por tanto en ese caso se utilizará la interpretación del grado de concordancia del coeficiente *Kappa-Fleiss*, mencionado anteriormente (Tabla 3). En caso de obtener un p-valor mayor que 0,05 no se puede rechazar la hipótesis nula, no poder rechazar que NO hay concordancia podría interpretarse como que no hay acuerdo entre las herramientas, pero realmente lo que se tiene es que el test no es significativo.

Los resultados obtenidos se resumen en la Tabla 4. Como se puede ver, los resultados del test para GC y FE son significativos, es decir, en ambos casos, los p-valores son menores que el umbral establecido por lo que se puede rechazar tanto H_0^{GC} como H_0^{FE}. Así que podemos admitir que hay concordancia entre las herramientas al detectar dichos *Design Smells* y se aplica la interpretación del coeficiente *Kappa-Fleiss* resultando en que en ambos casos el acuerdo es **pobre**.

RQ2 ¿Cuáles de las herramientas seleccionadas coinciden más al detectar *Design Smells God Class* y *Feature Envy*?

Acuerdo entre pares		inCode	inFusion	iPlasma	JDeodorant	JSmellSensor
Together	GC	p : 0,528	p : 0,528	p : 0,528	p : 0,211	p : 1
	FE	p : 0,00937 K : 0,26 débil	p : 0,00937 K : 0,26 débil	p : 0,00937 K : 0,26 débil	p : 1	p : 0,561
InCode	GC		p : 0 K : 1 muy bueno	p : 0 K : 1 muy bueno	p : 0,965	p : 0,000442 K : 0,351 débil
	FE		p : 0 K : 1 muy bueno	p : 0 K : 1 muy bueno	p : 0,417	p : 0,757
InFusion	GC			p : 0 K : 1 muy bueno	p : 0,965	p : 0,000442 K : 0,351 débil
	FE			p : 0 K : 1 muy bueno	p : 0,417	p : 0,757
iPlasma	GC				p : 0,965	p : 0,000442 K : 0,351 débil
	FE				p : 0,417	p : 0,757
JDeodorant	GC					p : 0,211
	FE					p : 0,91

Tabla 5: Resumen del grado de concordancia entre pares de herramientas, al detectar GC y FE en el primer quasi-experimento.

Para analizar esta pregunta de investigación, hemos estudiado el grado de concordancia entre los pares posibles de las 6 herramientas ($C_{6,2} = 15$ comparaciones), al detectar tanto *God Class* como *Feature Envy*.

En la Tabla 5 se muestra el resumen del grado de concordancia al detectar tanto GC como FE. Se indican en gris los p-valores mayores que 0,05. En estos casos no se puede rechazar la hipótesis nula, los tests no son significativos, se puede suponer que no hay acuerdo entre evaluadores, que la coincidencia es peor que si se hubiera hecho al azar. En el resto de los casos, se rechaza la hipótesis nula y se asume que hay acuerdo entre los evaluadores teniendo en cuenta el valor de *Kappa-Fleiss* según la interpretación mostrada en la Tabla 3. Como cabía esperar, hay concordancia muy buena en el par inCode, inFusion ya que sus resultados eran idénticos en el primer resumen mostrado en la Tabla 2. A estas herramientas se une con un nivel de acuerdo muy bueno, el caso de iPlasma. Este resultado es interesante, pues la concordancia ha sido muy buena, aunque iPlasma ha detectado un número de clases con *Design Smells* diferente al detectado por el par (inCode, inFusion). La razón la encontramos en que iPlasma es una herramienta que está en el origen de inCode e inFusion. iPlasma fue desarrollada por el grupo de investigación LOOSE de la Universidad Politécnica de Timisoara en Rumanía, dirigido por el Dr. Radu Marinescu. Por su parte, inCode e inFusion son herramientas de la compañía intooitus con sede en Timisoara,

Romania que resulta ser una startup con origen en este grupo, y por tanto estas herramientas y las técnicas que aplican tienen su base en iPlasma.

En la detección de *God Class* se ha obtenido un acuerdo débil entre nuestra herramienta JSmellSensor con el grupo inCode, inFusion, iPlasma. Este acuerdo débil se puede explicar partiendo de que JSmellSensor es una herramienta de aprendizaje automático que parte de un conocimiento inicial tomando como expertos a las herramientas inCode y JDeodorant. Es significativo que los resultados de JSmellSensor después de introducir conocimiento extra para intentar mejorar el número de falsos positivos y falsos negativos, se haya separado completamente de JDeodorant pero se mantenga cerca de inCode (y grupo relacionado), aunque el grado de concordancia sea débil.

En cuanto al grado de concordancia entre pares de herramientas al detectar *Feature Envy* (Tabla 5), se mantiene el alto grado de concordancia entre inCode, inFusion e iPlasma. Sin embargo, en este caso se detecta un nivel de acuerdo débil entre Together y este grupo. El resto de herramientas se mantienen alejadas de este grupo y entre sí. En el caso de JSmellSensor es muy significativo que partiendo de JDeodorant y de inCode como expertos iniciales, una vez entrenado el algoritmo de aprendizaje, se distancia de ambos al detectar *Feature Envy*.

A partir de estos resultados, se decidió realizar nuevamente el estudio del acuerdo entre herramientas, sin tener en cuenta el grupo formado por inFusion, inCode, iPlasma. La evaluación de las tres herramientas JDeodorant, Together yJSmellSensor en conjunto se comparó y se obtuvieron unos p-valores de 0,782 y 0,95 para la detección de *God Class* y de *Feature Envy*, respectivamente. Esto indica que no se puede rechazar la hipótesis nula porque lo que se concluye que no hay acuerdo entre dichas herramientas en ninguno de los dos casos porque la coincidencia es peor que si se hubiera realizado al azar.

5. Descripción y análisis de una réplica

Dados los siguientes hechos: a) en los estudios realizados en el primer quasi-experimento se han obtenido varios p-valores que indican resultados no significativos, b) fue encontrado el grupo formado por iPlasma, inFusion, inCode como coincidente debido a tener un origen común, y c) el carácter experimental de nuestra herramienta JSmellSensor, se decidió realizar una réplica. En esta réplica que denominaremos segundo experimento, se seleccionó solamente iPlasma como representante del grupo de tres herramientas del mismo origen, se descartó el uso de JSmellSensor y se introdujeron las herramientas más citadas en el estado del arte actualmente: DÉCOR y PMD, quedando una selección de 5 herramientas formada por: Together, JDeodorant, iPlasma, PMD y DÉCOR. En la preparación de esta réplica se decidió trabajar con un amplio dataset. Este dataset está formado por todas las clases de 24 proyectos de código abierto escritos en Java obtenidos de SourceForge. En la Tabla 6 se muestran las características generales de este dataset.

En este segundo experimento se trata de mantener los mismos objetivos y preguntas de investigación que en el primer quasi-experimento. Sin embargo, dada la amplitud del dataset, en esta réplica se decidió estudiar únicamente la detección de *God Class*. En la Tabla 6 se muestra una descripción de los proyectos incluidos en el dataset. Estos proyectos se han elegido teniendo en cuenta introducir

Project name	Version	NoC	TLOCP
jAudio	1.0.4	416	117615
Freemind	1.0.1	782	106396
JasperReports	4.7.1	1797	350690
SQuirreL SQL Client	1.2	1138	71626
KeyStore Explorer	5.1	384	83144
DigiExtractor	2.5.2	80	15668
Angry IP Scanner	3.0	270	19965
Plugfy	0.6	28	2337
Matte	1.7	603	52067
sMeta	1.0.3	222	30843
xena	6.1.0	1975	61526
pmd	4.3.x	800	82885
checkstyle	6.2.0	277	41104
JDistlib	0.3.8	78	32081
JCLEC	4-base	311	37575
Java graphplan	1.0	50	1049
Mpxj	4.7	553	261971
Apeiron	2.92	62	8908
FullSync	0.10.2	169	24323
OmegaT	3.1.8	716	121909
Lucene	3.0.0	606	81611
Ganttproject	2.0.10	621	66540
JFreechart	1.0.X	499	206559
JHotDraw	5.2	151	17807

Tabla 6: Características del dataset indicando el proyecto utilizado, su versión, el número de clases y la cantidad total de líneas de código

proyectos de características diferentes en cuanto a tamaño, dominio y estado en el que se encuentra el proyecto. Toda la información recopilada forma parte de otros estudios realizados o en progreso actualmente. Se cuenta, entre otros datos, con los resultados de realizar la detección de *God Class* con las 5 herramientas seleccionadas. Este dataset está disponible para los investigadores en `https://gitlab.citius.usc.es/joseangel.taboada/ProjectNominalInformation`.

Cada herramienta ha detectado un número *God Class* muy diferente. Together ha detectado 159 clases, DÉCOR 321, PMD 559, iPlasma 564 y JDeodorant 1235. En cuanto a número de clases detectadas iPlasma y PMD se acercan pero sin embargo no coinciden en las clases detectadas. Dadas las dimensiones del dataset, para poder analizar las posibles coincidencias en la detección de *God Class* entre las diferentes herramientas, si las clases detectadas por unas herramientas están incluidas en las detectadas por otras, se ha optado por realizar un estudio basado en análisis de conceptos formales (FCA) [7]. En la Figura 1 se muestra el resultado de la construcción de un retículo de Galois a partir de un contexto donde los objetos son las clases y los atributos son las herramientas. El contexto se define indicando que un objeto (una clase del dataset) tiene un atributo por cada herramienta (e.g. PMD) que detecta a dicha clase como *God Class* (e.g. PMD detecta la clase como *God Class* entonces la clase se marca que tiene a PMD como atributo). La construcción de un retículo de Galois garantiza que un atributo se introduce una sola vez. Así por ejemplo en un nodo del

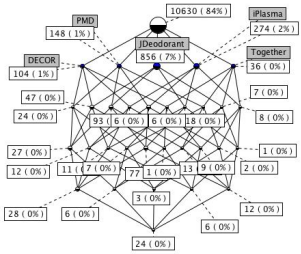


Figura 1: Retículo de Galois obtenido mediante análisis de conceptos formales (FCA).

retículo coincidirán todos los objetos (clases) que tienen los mismos atributos (todas las clases detectadas como *God Class* por las mismas herramientas) y se establece un orden jerárquico donde los nodos inferiores conectados con los superiores "heredan"los atributos y añaden nuevos (otras clases detectadas por nuevas herramientas). De esta forma, en el nodo completamente inferior estarían las clases detectadas por todas las herramientas (ninguna en este caso) y en el nodo superior, las clases que ninguna herramienta ha detectado como *God Class*.

En el retículo de Galois construido se aprecia en cada nodo en el nivel inferior las clases detectadas como *God Class* coincidentemente por las herramientas con las que se conecta en el nivel superior. En resumen se puede ver que solamente 24 clases son detectadas por todas las herramientas como *God Class* (nodo inferior cierre del retículo). Otros ejemplos de datos que se observan en el retículo son las clases que son detectadas solamente por una herramienta (en el primer nivel) como por ejemplo las 856 clases detectadas solamente por JDeodorant pero que no incluyen a las 104 clases solamente detectadas por DÉCOR, las 148 solamente detectadas por PMD, las 274 solamente detectadas por iPlasma y las 36 solamente detectadas por Together. También se aprecian otros datos como las 47 clases detectadas simultáneamente por DÉCOR y JDeodorant, etc.

Los resultados obtenidos del análisis de acuerdo entre evaluadores utilizando el estadístico *Kappa-Fleiss* se resumen en la Tabla 7. Como se puede ver hay un grado de concordancia pobre entre las herramientas al detectar *God Class*. Por otra parte, los resultados del estudio dos a dos entre las herramientas ($C_{5,2}$=10 comparaciones) se muestran en la Tabla 8. El mismo estudio se ha realizado aplicando *Cohen's Kappa* que es específico para 2 evaluadores y se han obtenido resultados muy iguales por lo que se ha optado por mantener el análisis utilizando el mismo coeficiente *Kappa-Fleiss* en todos los casos, tanto entre los 5 evaluadores conjuntamente como en el estudio 2 a 2.

Design Smell	p-value	*Kappa-Fleiss*	Interpretación
God Class (GC)	0	0.202	Acuerdo pobre

Tabla 7: Resultados del test Kappa-Fleiss, al comparar el grado de concordancia conjuntamente de las 5 herramientas seleccionadas en el segundo experimento.

Acuerdo entre pares	iPlasma	JDeodorant	PMD	DÉCOR
Together	p : 0 K : 0,156 pobre	p : $8{,}74e-08$ K : 0,0477 pobre	p : 0 K : 0,214 débil	p : 0 K : 0,244 débil
iPlasma		p : 0 K : 0,145 pobre	p : 0 K : 0,369 débil	p : 0 K : 0,195 pobre
JDeodorant			p : 0 K : 0,259 débil	p : 0 K : 0,14 pobre
PMD				p : 0 K : 0,272 pobre

Tabla 8: Resumen del grado de concordancia entre pares de herramientas, al detectar *God Class* en el segundo experimento.

6. Trabajos relacionados

Fontana et al [5] en el año 2011 describieron la experiencia de uso de varias herramientas de detección de *Design Smells*: JDeodorant, PMD, iPlasma, inFusion, Stench Blossom y DÉCOR. Realizaron una comparativa entre ellas según el lenguaje de programación que procesan, los *Design Smells* que detectan, si además realizan refactorings y cuestiones de usabilidad relativas a si se integra en un entorno de desarrollo o es una herramienta separada, si mantiene enlaces con el código en el que ha detectado los *Design Smells*, etc. Concluyeron que es muy difícil comparar herramientas porque no cubren los mismos *Design Smells*, la mayoría no tiene documentación, no se conocen las reglas que utilizan para detectar, ni los umbrales de las métricas que utilizan, si es el caso, etc. Todas las herramientas escogidas procesan código Java y se detectaron smells en diferentes versiones del mismo proyecto (Gantt Project). Aunque no utilizaron un estudio del grado de acuerdo entre evaluadores, muestran una comparación del número total de *Design Smells* detectados en aquellas que coinciden en detectar el mismo *Design Smell*. Por ejemplo, para *Feature Envy* comparan JDeodorant, Stench Blossom, inFusion, iPlasma, y para *God Class* comparan JDeodorant, inFusion e iPlasma. Los resultados en número son muy diferentes excepto entre inFusion e iPlasma en *God Class* que son idénticos aunque muy diferentes para *Feature Envy*.

También Fontana con otros autores [4] en el año 2012 realizaron una comparación entre cuatro herramientas de detección de *Design Smells*: JDeodorant, inFusion, PMD y Checkstyle. Con dichas herramientas analizaron el código de un

solo proyecto de código abierto GanttProject para detectar seis tipos de *Design Smells*, entre ellos *Feature Envy* y *God Class*. Comprobaron el acuerdo entre las herramientas. En este caso sí utilizaron el estadístico *Kappa-Fleiss*, encontrando que no hay acuerdo entre ellas y que en ello afecta, por una parte, la diferencia en las técnicas que emplean las herramientas para detectar y, por otra parte, el umbral de las métricas utilizadas en las herramientas cuando se emplean las métricas como técnica en la detección.

Hamid et al [8] en el año 2013 realizaron una comparativa entre las herramientas InCode y JDeodorant al detectar *God Class* y *Feature Envy*. Se trata de los mismos *Design Smells* que usamos en este trabajo pero únicamente usando dos herramientas. Se analiza el código de dos versiones de una aplicación multimedia Xtreme Media Player. Solamente comparan el número total de detecciones. No realizan un estudio de acuerdo entre evaluadores.

Recientemente Rasool y Arshad en [14] se ha publicado una comparativa entre herramientas de detección de *Code Smells* pero no incluye un análisis del grado de acuerdo entre evaluadores (en este caso las herramientas).

7. Conclusiones y trabajo futuro

De los resultados observados podemos concluir que hemos encontrado en el primer quasi-experimento dos grupos de herramientas claramente diferenciados: uno formado por inCode, inFusion e iPlasma y el otro por el resto de herramientas. En el primer grupo hay un grado de concordancia al detectar *God Class* y *Feature Envy* muy bueno, debido a que la procedencia de las herramientas es similar, y por tanto el proceso utilizado para detectar los *Design Smells*, tiene un diseño común. El otro grupo no tiene concordancia al detectar los *Design Smells God Class* y *Feature Envy*, confirmando así nuestras sospechas. No obstante, se ha realizado un segundo experimento con un mayor número de clases en cuyos análisis todos los resultados son significativos y en el que solamente aparece un representante del grupo coincidente en el anterior. En este estudio también se confirman las sospechas sobre el pobre o débil grado de acuerdo entre las herramientas al detectar aunque este estudio se ha centrado únicamente en la detección de *God Class*.

A la vista de los resultados observados, detectamos que sería necesario, por una parte disponer de herramientas capaces de detectar un amplio espectro de *Design Smells*. Por otra parte, sería necesario que dichas herramientas pudieran compararse fácilmente, de acuerdo a "patrones estándar" y se pudieran integrar en el proceso de software. Todo ello facilitaría que la industria adoptara algunas de esas herramientas, en forma similar a como ha adoptado las herramientas de *refactoring*. Se debe trabajar en la integración de estas herramientas con las tendencias actuales de automatización de proyectos, permitiendo obtener informes de detección de forma automática sin intervención humana.

Es posible que cada herramienta sea más sensible a determinadas características de los proyectos. Con esta hipótesis se elaborará un estudio para analizar algunas características de los proyectos que pueden influir en cómo detectan *Design Smells* las herramientas.

Referencias

1. Brown, W.J., Malveau, R.C., McCormick III, H.W., Mowbray, T.J.: AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. John Wiley and Sons (March 1998), http://www.antipatterns.com
2. El Emam, K.: Benchmarking kappa: Interrater agreement in software process assessments. Empirical Software Engineering 4, 113–133 (June 1999), http://portal.acm.org/citation.cfm?id=594380.594447
3. Fleiss, J.L.: Statistical methods for rates and proportions. New York: John Wiley, 2nd edn. (1981)
4. Fontana, F.A., Braione, P., Zanoni, M.: Automatic detection of bad smells in code: An experimental assessment. Journal of Object Technology 11(2), 5: 1–38 (2012)
5. Fontana, F.A., Mariani, E., Morniroli, A., Sormani, R., Tonello, A.: An experience report on using code smells detection tools. In: Fourth International IEEE Conference on Software Testing, Verification and Validation, ICST 2012, Berlin, Germany, 21-25 March, 2011, Workshop Proceedings. pp. 450–457 (2011)
6. Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: Refactoring: Improving the Design of Existing Code. Object Technology Series, Addison-Wesley (June 1999)
7. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer, Berlin/Heidelberg (1999)
8. Hamid, A., Ilyas, M., Hummayun, M., Nawaz, A.: A Comparative Study on Code Smell Detection Tools. International Journal of Advanced Science and Technology 60, 25–32 (2013)
9. López, C.: JSmellSensor, http://www.giro.infor.uva.es/clopeznozalPhD/clopeznozalPhD.html
10. López, C.: Detección de defectos de diseño mediante métricas de código. Ph.D. thesis, Universidad de Valladolid (dic 2012), http://www.giro.infor.uva.es/Publications/2012/Lop12b
11. López, C., Manso, E., Crespo, Y.: Evaluación de la eficiencia de métodos de identificación del defecto de diseño God Class. In: XVII Jornadas de Ingeniería del Software y de Bases de Datos (JISBD 2012). Universidad de Almería (2012), http://www.giro.infor.uva.es/Publications/2012/LMC12
12. Mendo Alonso, A., Sobrino Fernández, J., Pérez, J., Crespo, Y.: Evaluación de herramientas de detección y corrección de defectos de diseño. Tech. Rep. 2011/02, Grupo GIRO, Departamento de Informática, Universidad de Valladolid (March 2011), http://giro.infor.uva.es/Publications/2011/MSPC11
13. Pérez, J., López, C., Moha, N., Mens, T.: A classification framework and survey for design smell management. Tech. Rep. 2011/01, Grupo GIRO, Departamento de Informática, Universidad de Valladolid (March 2011), http://giro.infor.uva.es/Publications/2011/PLMM11
14. Rasool, G., Arshad, Z.: A review of code smell mining techniques. J. Softw. Evol. Process 27(11), 867–895 (Nov 2015)
15. Riel, A.J.: Object-Oriented Design Heuristics. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (April 1996)
16. Tiberghien, A., Moha, N., Mens, T., Mens, K.: Répertoire des défauts de conception. Tech. Rep. 1303, University of Montreal (2007)
17. Wake, W.C.: Refactoring Workbook. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2003)
18. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A.: Experimentation in Software Engineering: An Introduction, International Series in Software Engineering, vol. 6. Springer (2000)
19. Zhang, M., Hall, T., Baddoo, N.: Code bad smells: A review of current knowledge. J. Softw. Maint. Evol. 23(3), 179–202 (Apr 2011)

Calidad Ágil: Patrones de Diseño en un contexto de Desarrollo Dirigido por Pruebas

Manuel I. Capel[1]*, Anna C. Grimán[2], and Eladio Garví[1]

[1]Departamento de Lenguajes y Sistemas Informáticos, ETS Ingenierías Informática y Telecomunicación, Universidad de Granada, Granada, España.
{manuelcapel,egarvi}@ugr.es
[2]Departamento de Procesos y Sistemas, Universidad Simón Bolívar, Caracas, Venezuela.
agriman@usb.ve

Resumen Una de las prácticas más utilizadas dentro del desarrollo ágil, más específicamente en el Desarrollo dirigido por pruebas (TDD), son las pruebas unitarias, donde el marco de trabajo JUnit es muy utilizado actualmente. No todos los patrones y estilos disponibles pueden utilizarse en un contexto de pruebas unitarias como se plantea en TDD, ya que, para poder usarse, deben cumplir con un determinado conjunto de condiciones. Surge entonces la necesidad de conocer la factibilidad de que tales mecanismos puedan probarse a través de esta técnica.
En este trabajo se presenta una prueba de concepto de un proyecto de investigación en curso, realizándose un análisis de un conjunto reducido de patrones de diseño y estilos arquitectónicos para determinar su testeabilidad en un entorno de pruebas automáticas y TDD. Utilizando un caso de estudio, se consigue determinar la testeabilidad, así como los beneficios obtenidos en términos de calidad.

Palabras clave: Patrones de diseño; Desarrollo dirigido por pruebas; Calidad de Software; TDD.

1 Introducción

Algunos de los mitos que aún persisten en el paradigma del *desarrollo de software ágil* incluyen afirmaciones como las siguientes: la calidad del sistema se logra fácilmente con una arquitectura evolutiva, la adaptación a los cambios en los requerimientos es fácil, no hay que preocuparse por el *rendimiento*, la *escalabilidad*, la *seguridad*, la *usabilidad*, etc. del software hasta que su funcionalidad esté operativa. En la práctica, sin embargo, tales mitos quedan relegados y nos encontramos con la necesidad de obtener soluciones capaces de afrontar a nivel arquitectónico los compromisos típicos entre las características de calidad, tales como *usabilidad* frente a *seguridad*, *desempeño* frente a *mantenibilidad*, y *testeabilidad* frente a *rendimiento*, entre otras.

* Corresponding author

El Desarrollo Dirigido por las Pruebas (TDD, según su acrónimo inglés) surge inicialmente con la metodología XP [1] y ha cobrado cada vez mayor popularidad en el contexto de diferentes procesos o metodologías ágiles. TDD produce una arquitectura que surge de la no ambigüedad de las pruebas automatizadas.

Las arquitecturas ágiles son evolutivas, no se diseñan completamente antes de escribir el código. En general, se considera que con esta práctica se aumenta la calidad del software y se consigue un código altamente reutilizable, entre otros beneficios.

En la comunidad TDD existe una cierta desconfianza acerca de la utilización de patrones de diseño en el desarrollo de pruebas unitarias [2] que está retrasando su aceptación; sin embargo, los patrones de diseño sí son aceptados en el desarrollo de software en general.

Por otra parte, diferentes trabajos [3] [4] [5] [6] [7] [8] [9] [10] han abordado los beneficios del uso de patrones de diseño en cualquier actividad relacionada con el desarrollo de software. En [11] se propone una ontología para la mantenibilidad del software y, a partir de ella, se estudian los beneficios del uso de determinados mecanismos arquitectónicos en la mantenibilidad adaptados a los tipos de mantenimiento clásicos del software. En un trabajo posterior [12], se consideran 49 características, con sus correspondientes métricas, como base de la metodología DESMET, donde se usa el análisis de características para valorar métodos o herramientas de evaluación arquitectónica con el fin de asegurar la mantenibilidad de las arquitecturas de software.

En esta investigación presentamos una prueba de concepto realizada en el marco de un proyecto de investigación en curso. Nuestra propuesta puede aplicarse a cualquier marco de trabajo para pruebas unitarias de software. Se llevó a cabo un análisis con JUnit, un entorno de pruebas automáticas y TDD muy utilizado actualmente, de un conjunto reducido de patrones de diseño y estilos arquitectónicos para determinar su testeabilidad.

El objetivo de esta investigación a medio plazo es el análisis y caracterización de un conjunto de patrones de diseño útiles para TDD [13], que pueden ser empleados con efectividad en este contexto y, por consiguiente, dentro del marco de metodologías *ágiles* [14] [1] de desarrollo de software.

El resto del artículo se ha organizado como sigue, la sección 2 introduce la motivación que ha dado lugar a la investigación y se plantean los objetivos para la incorporación de patrones de diseño en las actividades relacionadas con TDD actualmente; se establecen las condiciones para la testeabilidad de patrones y se analiza la adecuación del conjunto seleccionado a ésta y otras características de calidad del software. En la sección 3 se comprueba, a través del caso de estudio SCACV (Sistema Automático de Control de Velocidad de un Vehículo), la testeabilidad de los patrones seleccionados. Finalmente, en la sección 4 se discuten los principales resultados obtenidos y se exponen las conclusiones del trabajo de investigación realizado.

2 Análisis de Patrones para TDD

Esta investigación supone el comienzo de un proyecto planificado para tres años cuyo objetivo fundamental es la realización de un análisis crítico, sistemático, repetible y apoyado en casos de estudio prácticos sobre las características de calidad según [15] de un conjunto de patrones y estilos de diseño arquitectónico.

En este trabajo sólo se ha analizado la testeabilidad, según un método de *pruebas unitarias*, de tres patrones que son muy utilizados actualmente en diseño y desarrollo de software, con el objetivo de promover su utilización en un entorno TDD para software complejo.

Una unidad de software ha de ser probada en su totalidad a través de la interfaz original, pero hay que tener en cuenta que cualquier estructura que se agregue por las clases que componen un patrón de diseño se convierte en una parte de la unidad de software que está siendo testeada. Por consiguiente, si como consecuencia de la utilización de un patrón, la unidad de software se convierte en más compleja, entonces será necesario testear más estados de dicha unidad. Se podría llegar incluso a la necesidad de testear un número infinito de estados si se producen dependencias de creación cíclicas. Con base en esta premisa y derivado de lo descrito anteriormente, surgen las siguientes preguntas:

1. ¿Es posible desarrollar código de tests que incorpore el uso de los patrones del catálogo GoF [16]?
2. ¿Qué beneficios se obtienen de la incorporación de los patrones al diseño evolutivo de la arquitectura obtenida a través de la práctica de TDD?
3. ¿Qué impacto negativo conlleva el uso de un patrón en el diseño obtenido a través de la práctica de TDD?

2.1 Condiciones de Testeabilidad de un Conjunto de Patrones de Diseño

Para responder a las preguntas planteadas anteriormente (2) y como prueba de concepto de nuestro proyecto, en primer lugar se realizó una selección de un pequeño conjunto inicial de patrones candidatos para TDD, basados en un conjunto de condiciones o criterios derivados de los conceptos y premisas de TDD, como son:

- El patrón debe poseer un enfoque funcional, por ejemplo, recibe o retorna uno o más valores.
- No deben existir dependencias de código ocultas que puedan producir efectos laterales en otros módulos. Si causa tales efectos, debe ser posible capturarlos y probarlos.
- No debe poseer un estado interno que haya de persistir tras la invocación de las operaciones o métodos de las clases que lo componen.
- No se pueden crear objetos dentro del código de métodos o constructores de las clases que pertenezcan al patrón.
- Debe ser posible verificar si durante la ejecución ocurrieron todos los eventos asociados al comportamiento y a la prueba unitaria de un patrón.

2.2 Evaluación de las Condiciones

Básicamente, las condiciones expresadas previamente convierten al patrón en un *componente software*, asegurando que cumplirá con la propiedad de *testeabilidad independiente*, es decir, el patrón encapsula todas las características que lo definen y no mantiene o crea por sí mismo dependencias con otros componentes de software que pudieran alterar su comportamiento dependiendo del contexto en que fuese utilizado. Afirmamos que un patrón ha de poder probarse independientemente y se ha de desplegar completamente para ser utilizado correctamente. Por tanto, el patrón visto como componente posee un comportamiento similar al de una función matemática, pudiendo aceptar valores, como datos de entrada, durante su utilización o parametrizarse, pero nunca va a enviar valores que puedan cambiar el estado de su entorno o crear una dependencia con el contexto, que haga visible su estado interno. Obviamente, lo anterior necesariamente implica que los métodos de las clases del patrón no pueden crear objetos de clases externas porque se estaría permitiendo la visibilidad de parte del estado del patrón, creando referencias accesibles fuera del contexto del patrón.

En las tablas de características se puede observar (en *itálica*) aquellas cualidades que obligatoriamente ha de cumplir un patrón para poder ser probado, según las condiciones de testeabilidad establecidas anteriormente.

Tabla 1. Características de calidad promovidas e inhibidas por el patrón *Observador*.

Característica	Sub-característica	Impacto	Cualidad
Mantenibilidad	Facilidad de cambio		Cohesión *Encapsulación* *Dinamicidad*
	Facilidad Análisis		Reusabilidad Complejidad
	Testeabilidad	-	Complejidad *Independencia*
Eficiencia	Tiempo respuesta		Rendimiento
	Uso recursos		
Funcionalidad	No acoplamiento	-	Complejidad
			Extensibilidad
Confiabilidad	Tolerancia a fallos	-	Recuperabilidad Exclusión mutua
		-	Monitorización Control
	Madurez		Previsibilidad *Sincronización*

2.3 Resumen de Patrones y sus Beneficios

Los patrones que se han utilizado para realizar este estudio: *Observador*, *Factoría Abstracta* y *MVC* cumplen todas las condiciones de testeabilidad anteriores. En el *Observador* se transmite automáticamente el cambio de estado desde los observables a los observadores, provocándose la ejecución del método *actualizar()* de estos, incluidos en la lista de cada observable. Tal método podría tener parámetros pero sólo es ejecutado por el observable, el cuál en todo momento

mantiene su estado oculto al resto. Tampoco es necesario crear referencias a ningún objeto observable porque no es necesario acceder a su estado; el propio observable se encarga de llamar a los métodos necesarios en los observadores sin que se rompa jamás su encapsulación.

Como se muestra en la tabla 1, la característica de *confiabilidad* puede verse inhibida con el patrón *Observador*, ya que este patrón no propicia la *recuperabilidad* ni la facilidad de monitorización del software. Un cambio menor sobre un objeto observable puede causar actualizaciones en cascada en los objetos observadores y sus objetos dependientes. También puede provocar que el rastreo de los errores sea mucho más difícil y, debido a que no se conoce su origen, las operaciones de recuperación pueden resultar un trabajo duro y difícil perjudicando la cualidad de *recuperabilidad*.

En el caso del patrón *Factoría Abstracta*, no se crean objetos dentro de métodos de clases del marco de trabajo, sino que éstas delegan la creación de los objetos que necesitan a las clases factoría, que incluirán métodos específicos con llamadas a los métodos de creación de los objetos-producto que se han de fabricar. Por tanto, en ningún momento se crean dependencias, vía referencias a nuevos objetos creados, desde dentro del patrón hacia componentes software externos al patrón.

La clase *Factoría Abstracta* puede contener un método parametrizado para invocar a métodos factoría concretos, pero estaríamos de acuerdo con la primera condición de testeabilidad, ya que el patrón mantendría un comportamiento funcional invariante en todo momento, pues sólo recibe el valor de un parámetro para seleccionar el método adecuado.

Tabla 2. Características de calidad promovidas e inhibidas por el patrón *Factoría Abstracta*.

Característica	Sub-característica	Impacto	Cualidad
Mantenibilidad	Facilidad de cambio	–	Flexibilidad
		|	*Encapsulación*
		|	*Dinamicidad*
	Facilidad Análisis	|	Reusabilidad
		-	Complejidad
	Testeabilidad	-	Complejidad
		|	*Independencia*
Eficiencia	Tiempo respuesta	-	Rendimiento
	Uso recursos		
Funcionalidad	No acoplamiento	-	Complejidad
		|	*Extensibilidad*
Confiabilidad	Tolerancia a fallos	|	Recuperabilidad
		|	Exclusión mutua
		-	Monitorización
		|	Control
	Madurez	-	Previsibilidad
		|	*Sincronización*

Tal como se intenta reflejar en la tabla 2, el uso de este patrón induce una considerable redundancia en el código y resultan afectadas determinadas cualidades del software, ya que el código se hace innecesariamente más complejo, lo

que planteará nuevos problemas en las fases de gestión, configuración posterior y hace más difícil la monitorización del software.

El rendimiento y la mantenibilidad de este patrón resultan perjudicados porque la duplicación de código inducida suele provocar bastantes problemas en la construcción y posterior mantenimiento del software. Los cambios que se realicen en el futuro, así como cualquier corrección de errores o fallas que pudieran producirse, afectarán a todas las variantes del mismo código.

Además, se trata de una solución poco flexible porque el introducir una nueva clase de productos hará necesario cambiar las clases factoría y todos sus descendientes.

Por último, el patrón *MVC* es realmente un compendio arquitectónico de varios patrones de diseño, que cumplen todos con las condiciones de testeabilidad de los patrones. Los modelos incluidos en este patrón no mantienen ninguna dependencia con la Interfaz de usuario (IU), ya que la separación entre la Vista y el Modelo que define el propio *MVC* se encarga de que esto sea siempre así. Por supuesto, todo va a depender de cómo programemos nuestro Modelo, si lo mantenemos independiente del resto de componentes y sólo se accede a su cambio de estado a través del Controlador (subcaracterística *no acoplamiento* de la tabla 3), entonces se puede comprobar trivialmente que las condiciones se satisfacen.

Tabla 3. Características de Calidad Promovidas e Inhibidas por el patrón *MVC*.

Característica	Sub-característica	Impacto	Cualidad
Mantenibilidad	Facilidad de cambio	-	Cohesión
			Encapsulación
			Dinamicidad
	Facilidad Análisis	-	Reusabilidad
			Complejidad
	Testeabilidad		Complejidad
		-	*Independencia*
Eficiencia	Tiempo respuesta	|	Rendimiento
	Uso recursos		
Funcionalidad	No acoplamiento	?	Complejidad
		?	*Extensibilidad*
Confiabilidad	Tolerancia a fallos	|	Recuperabilidad
		|	Exclusión mutua
		|	Monitorización
			Control
	Madurez	|	Previsibilidad
		|	*Sincronización*

La cualidad de *independencia* de la tabla 3 se refiere a la separación implícita de Vista y Modelo asumida por este patrón para realizar las pruebas, que afecta a la testeabilidad de la aplicación porque la Vista resulta difícil de probar de forma aislada, ya que se necesita de las acciones que ocurren en el Modelo.

3 Caso de Estudio: SCACV

Se ha desarrollado un applet en Java que simula un vehículo y su control automático de velocidad, ver http://lsi.ugr.es/~ist/Documentos/practica1.

Para iniciar su funcionamiento hay que pulsar el botón *Arrancar*, entonces se activa la palanca, que controla los distintos estados de control (*acelerando*, *frenando*, *activado*, *desactivado*, *reiniciando*).

Se ha programado con Swing/Java y los movimientos de la palanca se simulan accionando los distintos botones, Fig. 1(a). La palanca se moverá siempre que sea una transición permitida, Fig. 1(b). El botón de freno puede pulsarse en cualquier momento simulando el pedal del vehículo, mientras se mantenga pulsado el coche frenará y dejará de hacerlo en cuanto se suelte. Para apagar el motor y para repostar es necesario que el control automático se encuentre apagado y la velocidad del vehículo sea nula.

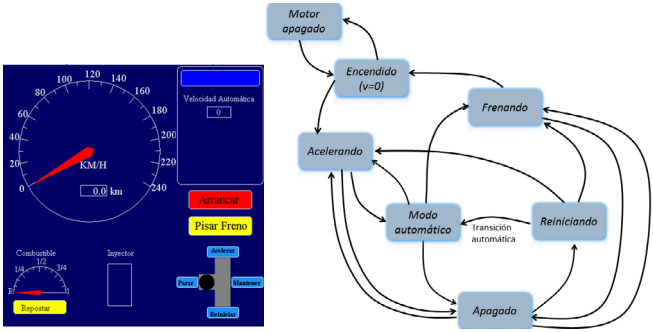


Fig. 1. (a) Simulación del Sistema Automático de Control de Velocidad de un Vehículo (SCACV). (b) Transiciones entre estados de control del vehículo y de la palanca.

Para desarrollar el (SCACV) se precisaron 3 paquetes diferenciados que contenían las clases (*código producido*): *ControlVelocidad*, *Monitorizacion* e *Interfaz* y los *Test Suite* que activan la ejecución de todos los casos de prueba (*Test Cases*). En la Fig. 2 se observa el diseño arquitectónico del SCACV propuesto, que muestra la utilización de dos patrones: *Observador* y *MVC*. El patrón de diseño *Factoría Abstracta* es parte del patrón *MVC* (cf. [16], pp.5-6) y se discutirá en la prueba unitaria de los paquetes *Interfaz* (Vista) y *Simulacion* (Controlador).

3.1 Pruebas Unitarias de los Patrones de Diseño

Para desarrollar un proyecto de pruebas con JUnit para determinados patrones de diseño cuya prueba consiste en verificar si durante la ejecución ocurrieron todos eventos de una lista, podemos basarnos en el *patrón de pruebas* denominado *Escuchador de Eventos*. Por ejemplo, el test del patrón *Observador* se realiza comprobando que cada vez que se dan las condiciones durante la ejecución, el

observable notifica a todos los *observadores* de su lista y como consecuencia cada uno de estos ejecuta su método *actualizar()*.
Se ha de crear una clase *Escuchador* que implemente la interfaz *TestListener* de Java y también la interfaz *Observador* del patrón de diseño del mismo nombre. La primera interfaz se utiliza para recibir *eventos textuales* que ocurren en las aplicaciones. Una clase Java que esté interesada en conocer la ocurrencia de un determinado evento textual implementará esta interfaz.

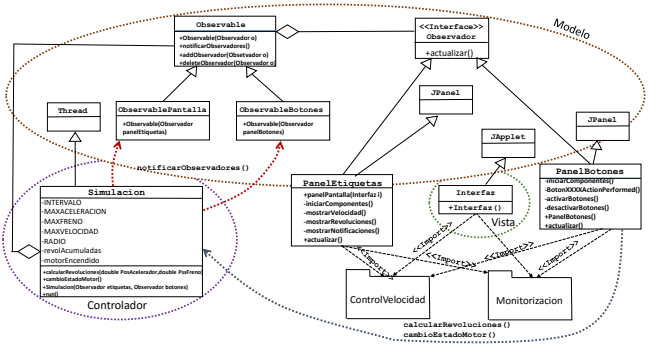


Fig. 2. Diseño arquitectónico del SCACV.

Un objeto de esta clase *Escuchador* se registrará en una lista interna del componente de la aplicación que puede modificar su texto llamando al método `addTextListener()`, de tal forma que cuando cambie el texto del componente, entonces se llama automáticamente al método `textValueChanged()` del objeto escuchador.

Para realizar las pruebas unitarias del SCACV, hemos incluido en tres paquetes diferenciados las clases de prueba para los paquetes correspondientes del código producido: *ControlVelocidad*, *Monitorizacion* e *Interfaz*, según el diseño arquitectónico propuesto en la Fig. 2. Cada uno de los paquetes de pruebas contiene un *Test Suite* que inicia la ejecución de todos los casos de prueba (*Test Cases*) de ese paquete. En el paquete *Interfaz* se incluyeron los casos de prueba para las clases: `Interfaz`, `ListaObservadoresObservables`, `Observable`, `PanelBotones` y `Simulacion`. No es necesario crear pruebas unitarias para la clase `PanelEtiquetas` ni para las clases que muestran indicadores en la interfaz de usuario, ya que su funcionamiento se basa en una modificación de la clase test para la propia `Interfaz`.

Lo anterior permite presentar los resultados obtenidos para los casos de prueba de las clases de código producido para el SCACV utilizando varios patrones de diseño de la siguiente manera:

1. Se realizó la implementación de la prueba de un escuchador de eventos que implementa *TestListener* de `junit.framework.*` y sirve para desarrollar las pruebas unitarias del patrón Observador utilizado en el paquete *ControlVelocidad*.
2. En el paquete *ControlVelocidad* se incluyó la descripción de los casos de prueba de las clases más relevantes, incluyendo clases de *prueba abstracta*, que contienen los métodos de la interfaz de los dispositivos, por ejemplo, *Pedal*:

```
public interface Pedal{
  public double leerEstado();
  public void soltar();
  public void actualizar();
}
```

3. No se implementaron casos de prueba para clases del tipo *Almacen*, ya que sus métodos son de tipo *getter()* y *setter()* y sus pruebas son triviales.
4. Por último, en el paquete de *Monitorización* se incluye una descripción de los casos de prueba para todas las clases que manejan los controles.

```
public class ObservadorTestListener
          implements TestListener, \emph{\emph{Observador}} {
 private List<List<Object>> eventos;
 public ObservadorTestListener() {   //Constructor
     eventos = new ArrayList<List<Object>>();
 }
 @Override
 public boolean actualizar() {
     eventos.add(nuevoObjetoEventoObservable());
     return true;
 }
 @Override
 public void startTest(Test test) {
     eventos.add(nuevoObjetoEventoInicioTest(test));
 } ...
 public List<Object> nuevoObjetoEventoObservable() {
     return Arrays.asList(new Object[ ]{"actualizar"});
 }
 public List<Object> nuevoObjetoEventoInicioTest(Test test){
     return Arrays.asList(new Object[]{"startTest",test});
 }...
 //Devuelve la lista de eventos acumulados
 public List<List<Object>> listaEventos() {
    return eventos;}
 }
}
```

Fig. 3. Implementación de la clase *ObservadorTestListener*.

Prueba del patrón *Observador* De acuerdo con el patrón de pruebas *Escuchador de Eventos*, se ha creado la clase *ObservadorTestListener* (Fig. 3), dentro del paquete *Simulacion*, como escuchador de eventos que se produzcan en la aplicación, que utiliza el patrón de diseño *Observador*. Esta clase cuenta con una lista de eventos que se va completando con objetos que se generan cada vez que se recibe un evento y que identifican al evento en cuestión utilizando, entre otros elementos, una cadena de caracteres. Implementa cada uno de los métodos de la interfaz *TestListener*, que se invocan cuando se produce un evento de *inicio de test*, *fin de test*, *error* o *fallo*, en la aplicación. Para cada uno de estos eventos se inserta un objeto diferente en la lista. El método *actualizar* de la Fig. 3 de la interfaz *Observador* también está implementado de forma que se inserta un objeto identificativo en la lista de eventos cada vez que es invocado durante la ejecución del test en `notificarObservadores()` del *Observable*, Fig. 4.

```
@Test
public void testNotificarObservadores() {
 testAniadirObservador();
 observable.notificarObservadores();
 List<List<Object>>esperados= new ArrayList<List<Object>>();
 esperados.add(observador.nuevoObjetoEventoInicioTest(this));
 esperados.add(observador.nuevoObjetoEventoObservable());
 assertEquals(esperados, observador.listaEventos() );
}
```

Fig. 4. Implementación de *testNotificarObservadores()*.

Por otra parte, se crea la clase `ListaObservadores/Observables` que contiene los casos de prueba de un *Observable*: `testNotificarObservadores()`, `testInsertaObservador()`, `testEliminaObservador()` y el creador de la *fixture*: `setUp()`. En esta clase se crea un objeto *ObservadorTestListener* y un objeto *TestResult*. Al objeto *TestResult* se le añade el objeto *ObservadorTestListener* como su escuchador. Posteriormente, al escribir el método de prueba `testNotificarObservadores()` (Fig. 4) se crea una lista con los objetos que se espera que contenga la lista de eventos, es decir: un objeto correspondiente al inicio del test, seguido de otro correspondiente a la llamada al método `actualizar()` del observador.

Por último, se compara la lista de eventos de `testNotificarObservadores()` con la del objeto escuchador *ObservadorTestListener*. Si coinciden, entonces el patrón *Observador* de la aplicación ha pasado la prueba con éxito.

Prueba del patrón *Factoría Abstracta* Se necesita delegar la creación de objetos a una clase *Factoría Especializada* que implementa una interfaz genérica definida previamente. El objeto *factoría* recibe llamadas, con el propósito de la creación de instancias de subclases de unaclase abstracta predefinida.

El patrón `FactoriaAbstracta` consiste en un objeto que contiene múltiples *métodos factoría*, de tal forma que una clase delega la responsabilidad de la creación del objeto deseado a otra clase a través de una composición.

```
public abstract class Componente extends Thread{
  private ListaObservadoresObservables obs;
}
public abstract class Panel extends JPanel{
  private void iniciarComponentes();
  public void setCodeBase();
}
```

```
Interface FactoriaComponentes import Thread{
\\singletons
  Simulacion crearSimulacion(List<Panel> paneles);
  ControlVelocidad crearControlVelocidad();
  Monitorizacion crearMonitorizacion(List<Object>
      dispositivos);
}
```

Fig. 5. Factoría especializada para la prueba del patrón *Factoría Abstracta* del SCACV.

Como marco de trabajo para realizar la prueba unitaria del patrón Factoría Abstracta del SCACV se ha definido la clase `Interfaz`, que hace uso de las interfaces *FactoriaPaneles* y *FactoriaComponentes* (Fig. 5). En el test de la clase `Interfaz` se ha de comprobar que en el constructor se crean todos los objetos específicos necesarios para el correcto funcionamiento del patrón *Factoria Abstracta* aplicado en el SCACV, así como los relojes que dirigen la simulación y la monitorización de la aplicación de forma asíncrona entre sí. El código del caso de prueba comprobará lo siguiente:

1. Para el objeto `simulacion`, que controla el ritmo de actualización de la interfaz y de los dispositivos, aseguramos que se ha creado la instancia de *Simulacion*, que es en sí mismo un *thread* independiente dentro de la aplicación completa.
2. Se prueba que se ha creado el panel de botones y se le ha pasado la instancia (única) de la clase singleton *Simulacion*.
3. Se ha creado el panel de etiquetas y se le ha pasado la instancia de `Interfaz`.
4. Se ha creado el *control de velocidad* y éste ha creado su reloj independiente.
5. Por último, se obtiene la instancia de *Monitorizacion* que crea otro reloj para que actúe de forma asíncrona con el control de velocidad y la simulación.

Prueba del patrón *MVC* Se trata ahora de conseguir realizar pruebas unitarias a una *Interfaz* que, si no se toman las acciones necesarias, presentará una mezcla difícil de separar entre la *lógica de interacción del usuario* (IU) y la denominada *lógica de presentación* (LP) de la aplicación. La lógica IU se refiere

a la gestión de las distintas interacciones del usuario con la interfaz de la aplicación, tales como pulsar un botón. El problema en este caso consiste en que la lógica de IU está fuertemente acoplada al código de manejo; de hecho, la lógica IU, la LP y la de *transformación de datos* (TD) están empotradas juntas en el código del manejador y es difícil saber dónde comienza una y termina la otra. Para poder realizar adecuadamente las pruebas unitarias de la lógica de IU de una aplicación, tendremos previamente que probar lo siguiente:

1. Toda petición desde la parte del usuario se tratará a través del Controlador.
2. El Controlador contiene toda la lógica de IU.
3. El Controlador no ha de mantener ningún acoplamiento con la Vista.

Si se respetan las reglas anteriores, entonces es posible realizar la prueba unitaria de la lógica IU (Controlador) de una aplicación como el SCACV. Además, la Vista se puede también probar si se define un modelo intermedio entre la Vista y el Modelo. De esta forma, el Modelo solo contendrá datos y lógica que tengan que ver exclusivamente con el negocio, mientras que el denominado *VistaModelo* contendrá ahora los datos (anteriormente en Modelo) que tengan relación con la Vista. De esta forma, se elimina cualquier posible conexión directa entre los componentes Vista y Modelo del patrón. En consecuencia, la Vista se conecta a VistaModelo, que contendrá los siguientes elementos:

- Referencia al Modelo
- La lógica de TD para cada uno de los requisitos definidos en la Vista.
- La LP para cada uno de los requisitos en la Vista.

En el caso del diseño del SCACV, el patrón *MVC* se aplica de acuerdo con la siguiente estructura:

- Modelo: clases de la Fig. 2, excepto *Simulacion*, *Interfaz* y Paneles.
- Vista: *Interfaz* y Panel de Etiquetas.
- VistaModelo: Panel de Botones (Fig. 6).
- Controlador: clase *Simulacion*.

La capa intermedia o *VistaModelo*, que permite la prueba unitaria independiente de las clases: *Interfaz* y *Simulacion*, viene representada por el *PanelBotones* (Fig. 6). Efectivamente, el código de los escuchadores de eventos de la interfaz de la aplicación se encuentra encapsulado en la clase *PanelBotones* y separado respecto de cada uno de los controles gráficos (botones, texto para completar, etc.) que se le presentan al usuario a través de la interfaz. *PanelBotones* contiene toda la lógica de TD que tiene un reflejo en la presentación de la Vista, ya que:

- Captura todos los eventos que se originan en la *Interfaz* (pulsar botones movimientos de la palanca, acelerar, etc.)
- Convierte determinados cambios en los datos del modelo (revoluciones del eje, estado del motor) a datos específicos de la Vista (representación gráfica de la velocidad, motor apagado/encendido).

```
public class PanelBotones extends JPanel implements \emph{\
    emph{Observador}}{
public PanelBotones(ControlVelocidad c,Monitorizacion m){...}
private void iniciarComponentes(){
/*Boton de arrancar.*/
  ...
  arrancar.addMouseListener(new java.awt.event.MouseListener
      (){...});
...
  synchronized public boolean actualizar(){
   ...
  }
  public void aniadirSimulacion(Simulacion s){
               simulacion = s;
  }
}
```

Fig. 6. La capa VistaModelo representada por la clase *PanelBotones*.

Por tanto, para terminar las pruebas unitarias de la aplicación, sólo nos resta probar el elemento VistaModelo (*PanelBotones*) pues la parte fundamental de la prueba unitaria del Controlador (*Simulacion*) y de la Vista (*Interfaz*) se realizaron anteriormente con la programación de las clases test de los patrones *Observador* y *Factoria Abstracta*, respectivamente.

4 Conclusión

En esta investigación se seleccionó un conjunto inicial de patrones de diseño y arquitectónicos (Observador, *Factoría Abstracta* y *MVC*) y se realizó una prueba de concepto para demostrar que pueden ser utilizados en un contexto de TDD siempre que se satisfagan algunas condiciones de comprobabilidad (*testability*) (sección 2.1).

De forma general, para poder llevar a cabo la prueba unitaria de los patrones en el marco de trabajo *JUnit* se implementó el patrón de pruebas *Escuchador de Eventos*. La clase *Escuchador* implementa los métodos de la interfaz *TestListener* (*inicioTest()*, *finTest()*, *falloTest()*, *error()*) y los métodos de una clase *fachada* asociada al patrón que es relevante para la prueba. El método *addTextListener()* se utiliza para modificar el texto del componente de la aplicación que se quiere probar y llamar al método *textValueChanged()* del objeto escuchador.

Un objeto *escuchador* hace posible registrar en una lista de eventos la ocurrencia de eventos fundamentales para relacionarlos con el desarrollo de la prueba, así como las llamadas a métodos de la *fachada* de cada patrón durante la ejecución de la aplicación.

La implementación de esta técnica de prueba unitaria se resume como sigue:

1. Crear un objeto de la clase *Escuchador* que registra en una lista interna los eventos textuales que ocurren en los componentes de una aplicación.

2. Un componente de la aplicación crea una instancia de *TestResult* que recoge los resultados de ejecutar un test y crea su objeto *escuchador*.
3. El *test* pasará sin producir errores o fallos si al finalizar coincide el contenido de la lista interna de eventos esperados del componente con la lista de eventos registrados en la lista de su *escuchador*.

Además, se identificaron las siguientes limitaciones en cuanto a la testeabilidad de un patrón que incluya una interfaz de usuario (IU) interactiva:

- Realizar pruebas unitarias de lógica IU, lógica de TD y LP, a la vez, no es posible.
- Difícil separación entre el código del modelo, vista y su controlador.

En general, la comprobación de código que contenga gestores de eventos ("*event handlers*") está limitada por el hecho de que los eventos no pueden ser generados en el test de pruebas de forma real y hay que recurrir a mecanismos que activan el manejador ficticiamente ("*mock handlers*") o que registran textualmente la ocurrencia de eventos o llamadas a métodos para poder programar los *Test cases*. Además, para poder realizar adecuadamente las pruebas unitarias de la lógica de IU de una aplicación ha de poder garantizarse la separación total entre el código del modelo y el de la interfaz de usuario de la aplicación.

Como trabajo futuro se contempla trabajar en la identificación y análisis de características de calidad de un conjunto más amplio de patrones de diseño susceptibles de ser utilizados con las técnicas TDD.

Referencias

1. Beck K. XP Explained (1st Edition). Addison-Wesley Professional, (1999)
2. Test Driven Design And Patterns (Wiki). http://c2.com/cgi/wiki?TestDrivenDesignAndPatterns. Retrieved 2012-08-15.
3. Gomathy. Software pattern quality comportment in service–oriented architectures. European Scientific Journal March 2014 edition v.10,9, pp.:412–423 (2014)
4. Khaer A., Hashem M.M.A., Masud R.. On Use of Design Patterns in Empirical Assessment of Software Design Quality. Proceedings of the International Conference on Computer and Communication Engineering 2008, Kuala Lumpur, Malaysia, May 13-15 (2008)
5. Khan K.Y., Mustaqeem A., Maqsood M. Improvement in quality of software architecture via enhanced-pattern driven architecture (EPDA). International Journal of Information Technology and Computer Science, 2012, 12, 31-39. DOI: 10.5815/ijitcs.2012.12.03 (2012)
6. Medvidovic N., Taylor R.. Framework for Clasifying and Comparing Architecture Description Languages. Software Engineering (ESEC/FSE97), 1301, 7, 60–76 (1997)
7. Harrison N., Avgeriou P. Leveraging architecture patterns to satisfy quality attributes. ECSA 2007, LNCS 4758, pp. 263–270, 2007. Springer-Verlag Berlin–Heidelberg (2007)
8. Bode S., Riebish M.. Impact evaluation for quality-oriented architectural decisions regarding evolvability. M. Ali Babar and I. Gorton (Eds.): ECSA 2010, LNCS 6285, pp. 182–197. Springer-Verlag, Berlin-Heidelberg (2010)

9. Ozkaya I., Kazman R., Klein M.H., Quality-Attribute-Based Economic Valuation of Architectural Patterns. Software Engineering Institute. Technical Report: CMU/SEI-2007-TR-003 (2007)
10. Lung C.H., Bot S., Kalaichelvan K., Kazman R. An Approach to Software Architecture Analysis for Evolution and Reusability. Software Engineering Institute (1997)
11. Griman A., Chávez L., Pérez M., Mendoza L., Dominguez K. Towards a Maintainability Evaluation in Software Architectures. In 8th International Conference on Enterprise Information Systems - ICEIS . Vol. 1: 555 - 558 (2006)
12. Griman A., Pérez M., Mendoza L., Losavio F. Feature analysis for architectural evaluation methods. Journal of Systems and Software, v. 79, 6, pp.: 871–888 (2006)
13. E. Guerra. Fundamental test driven development step patterns. PLoP '12 Proceedings of the 19th Conference on Pattern Languages of Programs. ACM Internal Conference Proceeding Series, October (2012)
14. Meszaros G. XUnit test patterns: refactoring test code. Person Education (2007)
15. ISO/IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models http://www.iso.org/iso/catalogue_detail.htm?csnumber=35733. Retrieved 2012-01-12.
16. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: elements of reusable object–oriented software. Addison–Wesley (1994)
17. Blé–Jurado, C. et al. Diseño Ágil con TDD (primera Edición) www.iExpertos.com. El libro se ha publicado bajo la Licencia Creative Commons. Enero (2010)

Hacia un entorno extensible basado en ADM para la refactorización de sistemas heredados

Abel Lorente Ramírez, Ignacio García-Rodríguez de Guzmán, Mario Piattini

Instituto de Tecnologías y Sistemas de Información (ITSI)
Universidad de Castilla-La Mancha
Paseo de la Universidad, nº4, 13071
{Abel.Lorente, Ignacio.GRodriguez, Mario.Piattini}@uclm.es

Resumen. Hoy en día siguen activos muchos sistemas heredados que presentan problemas que afectan a las distintas características de la calidad del software. Para mejorar estos problemas, existen herramientas de refactorización, cuyo objetivo es mejorar aspectos de calidad concretos sin afectar al comportamiento del sistema heredado. ADM (Modernización Dirigida por la Arquitectura), se presenta como el paradigma que basa el entendimiento y evolución de los sistemas software en MDA. Existen multitud de entornos que implementan estrategias de refactorización clásicas para mejorar la calidad de los sistemas, pero estas herramientas ofrecen un catálogo fijo de refactorizaciones. La propuesta que se presenta en este artículo consiste en un entorno flexible basado en ADM que permite la definición de *"bad-smells"* (clásicos y nuevos), aplicables a contextos concretos y su identificación en sistemas heredados, teniendo así una herramienta totalmente flexible y extensible.

1 Introducción

Una de las herramientas por excelencia para mejorar la calidad del software sin afectar a su funcionalidad es la reingeniería, más concretamente la fase de refactorización [1] que es el proceso de cambiar un sistema software de forma que mejore su estructura interna y aspectos de calidad sin cambiar su comportamiento exterior [2].

Aunque hoy en día pueden encontrarse multitud de herramientas que de un modo u otro asisten en el proceso de refactorización, existe una limitación común en todas ellas: no pueden ser extendidas, por lo que éstas sólo pueden actuar sobre el conjunto de *bad-smells* que tienen definido por defecto. Aunque es muy común encontrar un amplio conjunto de *bad-smells* [2] implementados para su detección, no se pueden extender los mecanismos para detectar nuevos *bad-smells* que lejos de estar catalogados, pueden considerarse como problemas de calidad en algún contexto concreto.

Como vía para soluciar este problema, se considera el paradigma MDA[1], que ha dado lugar a un nuevo enfoque de reingeniería denominado *Architecture-Driven Modernization* o ADM[2], que es el proceso de entendimiento y evolución de los sistemas

[1] http://www.omg.org/mda/
[2] http://www.omg.org/adm/

software existentes con el propósito de la mejora, modificación, interoperabilidad, refactorización, restructuración, reusabilidad, portabilidad, migración, traducción, SOA, y migración MDA del software, promovido por la OMG[3].

El metamodelo KDM (*Knowledge Discovery Metamodel*) es el principal pilar de ADM, y proporciona una visión integrada de la estructura del sistema heredado [3]. KDM puede verse como un conjunto de sub-metamodelos que permiten modelar cada una de las posibles vistas del sistema (código, interfaz de usuario, datos, aspectos arquitectónicos, etc.) pero manteniendo la coherencia y la trazabilidad entre los elementos de las mismas.

Por este motivo, en este trabajo se presenta una herramienta flexible y extensible basada en ADM que permite la definición de clásicos o nuevos *bad-smells* aplicables a contextos concretos y su identificación en sistemas heredados, con la finalidad de más adelante realizar, del mismo modo, refactorizaciones asociadas a esos *bad-smells*.

El artículo se organiza de la siguiente manera: la sección 2 ofrece una visión resumida de algunas de las propuestas más actuales para la refactorización de sistemas heredados; la sección 3 presenta el entorno de identificación de *bad-smells* y se detalla su arquitectura; y en la sección 4 se exponen las conclusiones finales del artículo.

2 Estado del arte

Actualmente existen diferentes herramientas de código abierto que realizan refactorizaciones. En la Tabla 1 se pueden observar las más relevantes.

Tabla 1. Comparativa de herramientas actuales de refactorización.

Herramienta	Refactorizaciones	Extensibilidad
Eclipse IDE[4]	Refactorizaciones entre las que se encuentran extraer o mover método, extraer clase o renombrados	No
BeneFactor[4]	Refactorizaciones básicas (extraer método, renombrar)	No
JDeodorant[5]	Mover método, extraer método, extraer clase, eliminar código duplicado	No
DNDRefactor[5]	Mover variables, métodos, clases entre paquetes o extraer método	No
AutoRefactor[6]	Entre 20 y 30 refactorizaciones sencillas, como eliminar modificadores, insertar sentencias if o simplificar expresión	No

[3] Object Management Group: http://www.omg.org
[4] https://eclipse.org/
[5] http://www.jdeodorant.org/
[6] https://marketplace.eclipse.org/content/autorefactor

Tras dicho análisis se extraen dos principales conclusiones en relación al tema que se desea tratar con este trabajo: (i) hay refactorizaciones que se encuentran prácticamente en la totalidad de las herramientas, y (ii) existe un aspecto que salta a la vista: ninguna de estas herramientas es flexible o extensible, pudiendo ser necesario el uso de herramientas distintas para resolver determinados problemas de calidad.

3 Entorno de identificación de *bad-smells*

Una de las principales motivaciones para el desarrollo de este framework de reingeniería es dotar al entorno de la flexibilidad necesaria para añadir nuevos mecanismos de detección de *bad-smells* y aplicación de refactorizaciones. Por este motivo, se ha planteado el desarrollo de este entorno como un *plug-in* Eclipse, por la facilidad que aporta a la hora de agregar nuevas funcionalidades a través de *plug-ins*. La Fig. 1 muestra un diagrama arquitectónico de la versión actual del entorno de refactorización.

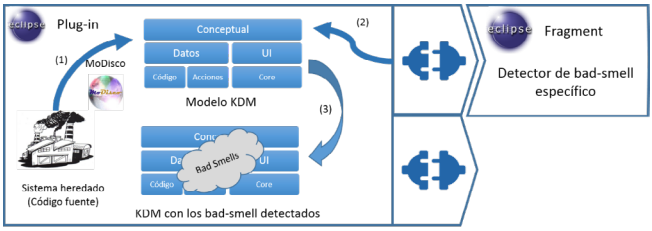


Fig. 1. Diagrama arquitectónico de la herramienta.

La estrategia que se ha seguido para conseguir que la arquitectura cumpla con estos requisitos radica en la modularidad, posibilitando el desarrollo de los detectores de bad-smells y las refactorizaciones como *plug-ins* del tipo "fragment" de Eclipse, que se asocian directamente con el núcleo de la herramienta (*plug-in host*).
La detección de *bad-smells* se lleva a cabo siguiendo los siguientes pasos: (i) ingeniería inversa y generación del modelo KDM del sistema heredado mediante el *plug-in* de MoDisco[7] integrado en el entorno de refactorización para hacer su uso trasparente al usuario (Fig.1. (1)); (ii) selección de *bad-smells* a detectar (cuyo "detector" o *plug-in* ha sido previamente seleccionado) sobre el KDM recuperado (Fig.1. (2)); (iii) generación de un nuevo modelo KDM marcado identificando las distintas ocurrencias del *bad-smell* en el modelo anterior (Fig.1. (3)).

Hasta el momento se ha desarrollado parte del entorno, y una de las cosas presentes es la consecución de integración de una primera versión de detector de *bad-smell* basado en el anti-patrón "Clase Dios" comprobando así la viabilidad de la propuesta.

[7] https://eclipse.org/MoDisco/

Conseguido este objetivo, se aprecia la diferencia con respecto a las alternativas existentes, ya que esta herramienta permite la incorporación de numerosas instancias de detectores personalizadas.

4 Conclusiones

En este artículo se han presentado los primeros pasos hacia un entorno para la detección de *bad-smells* en sistemas heredados. Este entorno pretende solucionar un problema del que adolecen las herramientas actuales de refactorización, que es la falta de flexibilidad para definir procedimientos de detección de problemas de calidad que no han sido clasificados como los clásicos *bad-smells*, y que podrían aplicarse a distintos ámbitos de la calidad del software. Otra de las ventajas es la posibilidad de crear repositorios abiertos para que los desarrolladores puedan proponer sus propias implementaciones.

El objetivo final es el desarrollo de un entorno completo de refactorización, que no sólo (i) detecte los *bad-smells*, sino que (ii) permita resolverlos en el modelo KDM y el sistema heredado. Por este motivo, este artículo presenta una visión general de la primera parte del objetivo principal, la detección de los problemas de calidad que se describan en los detectores de *bad-smell* desarrollados como *plug-ins*.

5 Agradecimientos

Este trabajo ha sido financiado por los proyectos: GINSENG (TIN2015-70259-C2-1-R), SEQUOIA (TIN2015-63502-C3-1-R), (Ministerio de Economía y Competitividad y Fondo Europeo de Desarrollo Regional FEDER, y MOTERO (Consejería de Educación, Ciencia y Cultura de la JCCM, y FEDER, PEII11- 0399-9449)

6 Referencias

[1] E. J. Chikofsky and J. H. C. II, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Softw.,* vol. 7, pp. 13-17, 1990.

[2] M. Fowler and K. Beck, *Refactoring: Improving the Design of Existing Code*: Addison-Wesley, 1999.

[3] R. Pérez-Castillo, I. G.-R. d. Guzmán, and M. Piattini, "Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems," *Comput. Stand. Interfaces,* vol. 33, pp. 519-532, 2011.

[4] X. Ge and E. Murphy-Hill, "BeneFactor: a flexible refactoring tool for eclipse," presented at the Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, Portland, Oregon, USA, 2011.

[5] Y. Y. Lee, N. Chen, and R. E. Johnson, "Drag-and-drop refactoring: intuitive and efficient program transformation," presented at the Proceedings of the 2013 International Conference on Software Engineering, San Francisco, CA, USA, 2013.

Quality metrics for mutation testing with applications to WS-BPEL compositions

Antonia Estero-Botaro, Francisco Palomo-Lozano, Inmaculada Medina-Bulo, Juan José Domínguez-Jiménez, and Antonio Garcia-Dominguez

Departamento de Ingeniería Informática, Universidad de Cádiz,
Avda. de la Universidad de Cádiz 10, 11519 Puerto Real, Spain
{antonia.estero,francisco.palomo,inmaculada.medina}@uca.es
{juanjose.dominguez,antonio.garcia}@uca.es

Resumen La prueba de mutaciones es una técnica de prueba basada en fallos que ha sido aplicada de forma satisfactoria a diversos lenguajes. Sin embargo, esta técnica puede ser muy costosa, por lo que se han propuesto diversas soluciones que reducen el número de mutantes producidos para disminuir su coste. Esta reducción del coste puede conseguirse mediante el análisis de los operadores de mutación empleados, pero se requiere la definición de métricas especializadas. Hasta ahora se han propuesto diversas métricas, pero no resulta fácil evaluar su efectividad. Un paso más allá en la evaluación de las técnicas de reducción de mutantes sería disponer de una métrica más adecuada que permitiera determinar de forma objetiva la calidad de un conjunto de mutantes con respecto a un conjunto determinado de casos de prueba. Este trabajo introduce esta métrica, que se extiende de forma natural a los operadores de mutación y que puede ser utilizada para reducir el número de mutantes, en particular los mutantes equivalentes. Además, se presenta una herramienta de análisis de mutaciones firme para composiciones WS-BPEL así como los resultados experimentales obtenidos comparando diferentes métricas sobre varias composiciones.

Keywords: mutation testing, firm mutation, quality metrics, quality of mutation operators, service compositions, WS-BPEL

Evaluando la efectividad de un modelo abstracto de transacciones para la prueba de transacciones en servicios web

Rubén Casado [1], Javier Tuya [2], Muhammad Younas [3]

[1] Treelogic, [2] Universidad de Oviedo, [3] Brookes University, UK

Abstract. Las transacciones en servicios web son utilizadas para elaborar aplicaciones distribuidas en internet que de forma eficiente y fiable permiten acceder a múltiples usuarios de forma simultánea, Las investigaciones actuales desarrollan diversos modelos y protocolos para mejorar la fiabilidad y rendimiento de las transacciones en servicios web. Sin embargo, hay poca investigación relativa a la realización de pruebas de los diferentes modelos y protocolos de transacciones en servicios web. Este artículo presenta un modelo abstracto de transacciones de acuerdo con diferentes estándares de transacciones. El modelo propuesto es implementado como un prototipo que es evaluado utilizando un caso de estudio de Jboss Transaction. La evaluación muestra que el sistema propuesto tiene la capacidad de generar automáticamente casos de prueba y de detectar diversos fallos en las transacciones que se ejecutan bajo diferentes estándares de transacciones web.

Keywords: transacciones, servicios web, pruebas de software, detección de fallos, fiabilidad

Artículo publicado en:
Concurrency and Computation: Practice and Experience.
Volume 27, Issue 4, 25 March 2015, Pages 765–781
DOI: 10.1002/cpe.2851
Q2 in JCR 2014 (Computer Science, Software Engineering)

Desarrollo de Software Dirigido por Modelos

Desarrollo de Software Dirigido por Modelos

Coordinadores: Cristina Vicente Chicote y Juan de Lara

Xabier De Carlos, Goiuria Sagardui, Salvador Trujillo, Alain Perkaz, Mikel Cañizo y Aitziber Iglesias. *Evaluating Embedded Relational Databases for Large Model Persistence and Query*. (Completo)

Juan-Pablo Salazar-Álvarez, Elena Gómez-Martínez y Miguel de Miguel.
Performance Analysis of Persistence Technologies for Cloud Repositories of Models. (Completo)

Rubén Salado-Cid y José Raúl Romero. *Lenguaje específico para el modelado de flujos de trabajo aplicados a ciencia de datos*. (Completo)

César Cuevas, Patricia López Martínez y Jose M. Drake. *MDDE: Una concepción genérica para diseño de entornos de desarrollo de software basados en MDSE*. (Completo)

Alfonso de La Vega, Diego García-Saiz, Marta Zorrilla y Pablo Sanchez. *Desarrollo Eficiente de Lenguajes Específicos de Dominio para la Ejecución de Procesos de Minería de Datos*. (Completo)

Iván Ruiz-Rube, Tatiana Person y Juan Manuel Dodero. *Static analysis of textual models*. (Corto)

Juan Boubeta-Puig, Juan Hernández, Enrique Moguel, Juan Carlos Preciado y Fernando Sánchez-Figueroa. *Una Propuesta de Editor Gráfico para el Modelado y la Generación de Código de Patrones de Eventos sobre Drones*. (Corto)

Loli Burgueño, Manuel Wimmer y Antonio Vallecillo. *Towards Distributed Model Transformations with LinTra*. (Corto)

Javier Troya, Sergio Segura y Antonio Ruiz-Cortés. *Towards the Automation of Metamorphic Testing in Model Transformations*. (Corto)

Juan M. Vara, Valeria De Castro, David Granada y Esperanza Marcos.
Bringing together existing Business Modeling flavors. (Corto)

Jesús Sánchez Cuadrado, Esther Guerra y Juan De Lara.
Análisis de transformaciones ATL con AnATLyzer. (Demo)

Abel Gómez-Llana y José Meseguer.
Una herramienta para evaluar el rendimiento de aplicaciones intensivas en datos. (Demo)

Xavier Oriol, Ernest Teniente, Albert Tort: *Computing repairs for constraint violations in UML/OCL Conceptual schemas*. Data & Knowledge Engineering, 99 (2015) 39–58. (Relevante)

Hugo Bruneliere, Jordi Cabot, Javier Luis Cánovas Izquierdo, Leire Orue-Echevarria Arrieta, Oliver Strauss and Manuel Wimmer: *Software Modernization Revisited. Challenges and Prospects*, IEEE Computer 48(8): 76-80. 2015. (Relevante)

Evaluating Embedded Relational Databases for Large Model Persistence and Query

Xabier De Carlos[1], Goiuria Sagardui[2], Salvador Trujillo[1], Alain Perkaz[1], Mikel Cañizo[1], and Aitziber Iglesias[1]

[1] IK4-Ikerlan Research Center, P. J.M. Arizmendiarrieta, 2 20500 Arrasate, Spain
{xdecarlos,strujillo,aperkaz,mcanizo,aiglesias}@ikerlan.es
[2] Mondragon Unibertsitatea, Goiru 2, 20500 Arrasate, Spain
gsagardui@mondragon.edu

Abstract. Large models are increasingly used in Model Driven Development. Different studies have proved that XMI (default persistence in Eclipse Modelling Framework) has some limitations when operating with large models. To overcome them, recent approaches have used databases for the persistence of models. EDBM (Embedded DataBase for Models) is an approach for persisting models in an embedded relational database, providing scalable querying mechanism by runtime translation of model-level queries to SQL. In this paper, we present an evaluation of EDBM in terms of scalability with existing approaches. GraBaTs 2009 case study (models from 8.8MB to 646MB) is used for evaluation. EDBM is 70% faster than the compared approaches to persist XMI GraBats models into databases and executes the GraBats query faster, as well as having a low memory usage. These results indicate that an embedded relational database, combined with an scalable query mechanism provides a promising alternative for persisting and querying large models.

Keywords: Model-Driven Development, Large-Scale Models, Persistence, Query, Runtime Translation, Evaluation

1 Introduction

Automatizing and optimizing development processes is crucial to reduce development efforts and time to market of industrial projects, which are the main drivers of competitiveness. Model Driven Development (MDD) promises improvements in the development process through an intensive use of abstractions, specified by models. Models are considered first class entities during the development process, so that engineers may use them for different purposes such as code and documentation generation. Operating with models requires querying, editing and transforming them.

Eclipse Modelling Framework (EMF) is a mature and a widely used modelling framework provided within the Eclipse IDE, and XML Metadata Interchange (XMI) is the default mechanism to persist models. However, XMI entails memory and execution problems as the size of the model increases [1,2]. Thus,

in industrial domains (e.g. windpower or railway) where systems can comprise a large number of elements such as sensors, actuators and control units, using XMI is not an option. Consequently, effectively supporting such domains requires using additional persistence mechanisms. Recent approaches opt for the use of databases to persist and operate with models, for example Morsa [2] or Neo4Emf[1]. In previous works [3][4], we have presented Embedded DataBase for Models (EDBM), an alternative persistence mechanism for EMF models based on an embedded relational database. Model operations are also provided through an scalable solution to query them based on a runtime translation.

In this paper, we briefly introduce EDBM and evaluate it comparing with other persistence mechanisms. While technical description of our approach has been previously presented [4], this paper contributes with a detailed evaluation which validates its usefulness and scalability. The approach is evaluated using models of different sizes (from 8.8MB, containing 14 Java classes and 70447 model elements, to 646MB, containing 5984 Java classes and 4961779 model elements) extracted from the GraBaTs 2009 case study [5]. The experiments show that EDBM is able to query persisted models maintaining low memory footprint but also taking a reasonable execution time. We compare results (storage size, insertion time, memory usage and execution time) of our approach with XMI and other database-based persistence solutions.

The rest of the paper is organised as follows: Section 2 describes some background and motivation of this work. Section 3 reviews related work, and compares it with EDBM. Then, the approach is presented in Section 4 and it is evaluated in Section 5. This paper ends with conclusions and future work in Section 6.

2 Background and Motivation

In EMF, models are persisted by default using XMI, an XML-based information persistence format standardised by the OMG. Before operating with models, all the information persisted in the file has to be loaded in memory. Once the model is in memory, information is operated: editing, querying, generating code, executing transformations, etc. If the model is modified, information in memory has to be stored again in the XMI file. However, transferring the information from physical file to memory and vice-versa entails problems with large models [6]. To overcome these problems, most recent approaches opt to leverage database capabilities. Morsa[2], Neo4EMF[1], MongoEMF[7] or EMF Fragments[8] use databases for model persistence. Each approach provides useful mechanisms for operating with large-scale models: partial load of the information, loading the information on-demand, caching, etc.

The aforementioned approaches, fully delegate the physical storage of information in models to the underlying database management system. This motivates us to explore different alternatives for providing an efficient model persistence mechanism that could act as a drop-in replacement for XMI when working with large models. Besides persistence, scalability should also be provided when

operating models (e.g. querying, editing or executing transformations). Our aim is to provide a mechanism that supports (i) leverage database capabilities but using a file-level persistence mechanism; and (ii) operating models at a meaningful level of abstraction without needing to fully load them into memory. In this sense, our solution aims to provide a scalable mechanism for persisting and querying large-scale models. EDBM uses Epsilon Object Language (EOL), a model-level language that supports querying models, but can also be used to edit and transform models.

3 Related Work

We have classified the related approaches into two different groups: (i) model persistence and (ii) model query languages.

3.1 Model Persistence

Model persistence using relational and non-relational database management systems allows to load models on-demand, overcoming the memory problems of XMI. CDO[9] provides a repository, where models can be persisted using different database management systems (NoSQL and RDBMS). It also offers other features such as multi-user access, collaboration and concurrent model access of the repository or mechanisms for querying persisted models. Besides models, other information (metamodels, history) is also persisted together within the database, to provide model version control and sharing. However, scalability seems not to be the design goal of CDO [6]. Teneo[10] uses a relational database for persistence of models. It supports mapping between EMF objects and a relational database. The database schema can be metamodel-independent or metamodel-specific. Schemas are customized through metamodel annotations. Different evaluations show that Teneo does not scale well [11].

In a similar vein, Morsa[2] provides large-scale model persistence using MongoDB, a document-based NoSQL database back-end. The approach provides on-demand loading mechanisms and cache replacement policies that can be chosen by the end-user. This allows working with large models with a low memory footprint. Neo4EMF[1] is based on a transactional property-graph NoSQL database back-end (Neo4J). The approach supports the mapping between EMF models and Neo4J graphs. Neo4EMF provides different strategies for on-demand loading. Mongo EMF[7] is based on the document-based MongoDB. This approach provides an extensible and flexible framework based on OSGi declarative services. EMF Fragments[8] is a framework for persisting model fragments, and can be used with different NoSQL back-ends such as MongoDB, HBase or with distributed file systems. Each fragment contains different model elements and relations, and the fragmentation strategy is specified by the users at the metamodel level. Then, for querying models, only required fragments are loaded in memory. The Mondo Project[12] aims to provide an scalable MDD environment, which includes persistence mechanisms based on databases. In [11], the

authors present two prototypes that use a NoSQL database for persistence of models (Neo4J and OrientDB) and compares benchmark of these prototypes with benchmarks obtained from models persisted in a XMI file and in a relational database. In [13], the authors present a framework and a methodology for benchmarking persistence of models on different NoSQL stores. To the best of our knowledge, approaches based on different NoSQL back-ends (suitable for distributed databases and large quantities of data (a.k.a. Big Data)) overcome the memory problems of XMI while relational database approaches already have scalability problems. We opt for a persistence based approach on a relational and embedded database, since it facilitates integration of the persistence at the same-level of XMI (file-level) but leveraging the capabilities of a database. Additionally, we base our work on the hypothesis that using a relational embedded database with a metamodel-agnostic data schema overcomes the memory problems of XMI. On the contrary to other relational database proposals we persist a single model in the database.

3.2 Model Query Languages

Languages for querying models are useful when specifying rules that obtain elements from models (e.g. all elements satisfying a condition). Some approaches provide model-level languages closer to modelling engineers: Object Constraint Language (OCL), EMF Query[14], IncQuery[15] or EOL[16]. EOL also provides support for specifying query expressions to modify models.

Other approaches propose persistence specific and dependent languages that leverage capabilities of the persistence (optimising queries): COCL (a.k.a. CDO-OCL[3]), for models persisted using CDO; MorsaQL [2] for models persisted using Morsa; database specific languages such as SQL or Cypher. In [2], the authors describe benchmarks of different query languages (OCL, MorsaQL, EMF Query, etc.) executed over models persisted using XMI, Morsa and CDO.

There are also some proposals to generate persistence level queries from model-level queries. In [17], the authors describe an approach focused on generating MySQL code from a given OCL expressions. An approach to generate SQL queries from OCL invariants is presented in [18]. In [19], the authors describe an approach that generates views using OCL constraints, and then uses them to check the integrity of the persisted data. This approach has been implemented in OCL2SQL[4], a tool that generates SQL queries from OCL constraints. A similar approach for integrity checking is proposed in [20]. Another approach described in [21] details a method that executes queries in persistence-level (SPARQL) and the results are the input of model-level queries (OCL).

EDBM provides a solution able to query models using a model-level query language (EOL) with the efficiency of SQL to query models persisted in a database . While existing approaches translate OCL constraints into SQL queries at compile-time, our approach generates SQL queries from OCL-like expressions

[3] More info at "https://wiki.eclipse.org/CDOQuery_OCL"
[4] Read more at "http://dresden-ocl.sourceforge.net/usage/ocl22sql/"

at runtime. We have based our work on [22], which describes an approach where EOL is used to query large datasets stored on relational databases composed by one table. While in this approach naive translation provided by Epsilon Model Connectivity Layer (EMC) is used to query information persisted in a single-table database, EDBM-Query provides custom translations of SQL queries that leverage persistence capabilities when querying models persisted in an embedded database.

4 Embedded DataBase for Models (EDBM)

Embedded DataBase for Models (EDBM) is an approach that is focused on the scalable persistence of models, but also on the scalable operation of models through a querying mechanism that leverage database capabilities.

4.1 EDBM: Scalable Persistence

EDBM follows the main principle of using an embedded relational database for persisting the models. In this sense, a survey has been performed to identify systems that provide support for embedded relational databases in Java. The survey revealed that currently, SQLite[5] and H2[6] are the most mature options. After performing some preliminary tests and consulting existing benchmarks[7], we concluded that H2 has a better performance profile.

The approach uses a metamodel-agnostic schema for model persistence. This way, EDBM supports the persistence of models that conform to arbitrary metamodels and does not require to modify the schema when the domain metamodel evolves (only stored information has to be updated). Figure 1a illustrates the metamodel-agnostic schema used by EDBM and it is described next:

- **Metamodel information:** *Class* and *Feature* tables are used to store each meta-class and each structural feature existing in the domain metamodel. With these tables, the approach is able to know metamodel-related information, but using a domain-agnostic data schema. An unique id (`ClassID`, `FeatureID`) is used to identify classes and features.
- **Model elements:** `Object` table is used to persist all the elements of the model. An unique id (`ObjectID`) and the id of the meta-class (`ClassID`) is stored for each model element.
- **Structural features:** `AttributeValue` and `ReferenceValue` tables are used to store feature values of each model element. `ObjectID` and `FeatureID` are used to identify each value (`Value` column). In case of attributes, the value contains a primitive value. And in case of references, id of the referenced value and its meta-class id are stored (`Value` and `ClassID` columns).

[5] `https://sqlite.org`
[6] `http://www.h2database.com`
[7] SQLite vs. H2 comparative at "http://tinyurl.com/puxdllm"

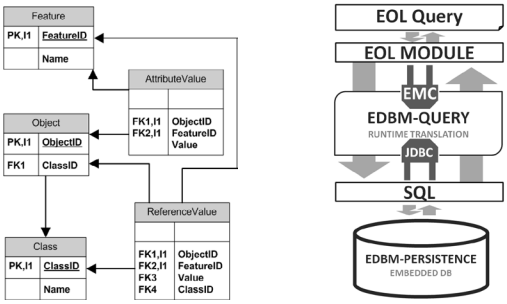


Fig. 1: (a) Database-schema of EDBM, and (b) overview of the approach.

4.2 EDBM: Scalable Model-Level Query

Using EDBM, models are persisted in an embedded relational database. In this case, leveraging SQL is more efficient than naive iteration. However, model-level queries are focused on interaction with models (and are closer to modelling engineers). Being so, use of model-level queries is more appropriate than exposing modelling engineers to the database directly with a persistence-level language (SQL).

EDBM allows to execute queries expressed in a model-level query language (EOL) and translate then in runtime to a persistence-level query language (SQL). Reasons for EOL[16] selection include: (i) it is an OCL-based language that also provides features of imperative languages such as the use of variables on the queries, query expressions for model modification and the specification of query expression chains; and (ii)it is the base of other model-specific languages like Epsilon Transformation Language (ETL) or Epsilon Generation Language (EGL) for model-to-model or model-to-text transformations, etc.

EOL and SQL are query languages that do not have direct mapping, since EOL provides constructs that are not provided by SQL (e.g. variables). In this situation, translation from EOL to SQL can be performed in two ways: at compilation-time or at runtime. Compilation-time translation requires performing a static analysis of the queries and reordering them before the translation. By contrast, if the translation is performed at runtime, query expressions are translated one by one, and they are executed against the database only when the results are required by a following expression. Therefore, we have opted to execute translation at runtime.

Figure 1b illustrates an overview of EDBM. As shown in the figure, EOL Module is the responsible for parsing and executing queries expressed using

Table 1: Execution environment of each evaluation.

	NoSQL_Eval[11]	**Morsa_Eval[2]**	**EDBM_Eval**
Processor	Intel Core I5-2300 @ 2.80 GHz	Intel Core I7-260 @ 3.70 GHz	Intel Core I7-3520M @ 2.90 GHz
Physical Memory	8GB	8GB	8GB
Operative System	Windows 7 (64-Bit)	Fedora Core 17 (64-Bit)	Windows 7 SP1 (64-Bit)
JVM	Java SE v1.6.0	OpenJDK JVM 1.7	Java SE v1.8.0
Eval. Persistence	XMI, CDO, Neo4J, OrientDB	XMI, CDO, Morsa	XMI, EDBM
Eval. Query Leng.	EOL	Plain EMF, OCL, COCL, EMFQuery, IncQuery, MorsaQL	Plain EMF, EOL
Query Repetitions	20	2	100

EOL and the EMC provides different interfaces that make possible to connect and communicate with the EOL Module. In this sense, EDBM implements such interfaces to provide the connection with the EOL queries. To be able to connect and execute queries against the database, EDBM uses the JDBC driver of H2. More details of how each artifact is used during the translation, and sample translation and execution of an EOL query had been provided in a previous paper [4]. At this point of development, EDBM supports translation of read-only EOL expressions (e.g. queries containing query expressions such as selects, collects, etc.). However, we are already working to add the support for expressions that modify models (e.g. query expressions for creating new element instances) in a future prototype of the approach.

5 Evaluation

This section presents the evaluation of persistence and query mechanisms provided by EDBM. The GraBaTs 2009 case study [5] has been selected, since it is widely used to evaluate model persistence and querying approaches.

The GraBaTs models [8] have been persisted using EDBM. These models specify source code of different Java packages and conform to the JDTAST metamodel which contains abstractions of the Java source code. The size of models ranges from 8.8MB, containing 14 java classes and 70447 model elements (set0), to 646MB, containing 5984 java classes and 4961779 model elements (set4).

We have evaluated EDBM and XMI applying the GraBaTs query to the models. GraBaTs query returns all the singleton classes of a model. The query has been expressed using EOL for EDBM, and EOL and Plain EMF for XMI.

To compare EDBM with existing approaches we have used the results of *NoSQL_Eval* [11] and *Morsa_Eval*[2], where alternative mechanisms for persisting and querying large-models are evaluated. Table 1 illustrates the execution environment of such studies. *NoSQL_Eval* evaluates XMI, CDO, Neo4J

[8] More information and resources available at "http://www.emn.fr/z-info/atlanmod/index.php/GraBaTs_2009_Case_Study"

Table 2: Time required to insert XMI model into database and storage size.

		XMI	Neo4J[11]	OrientDB[11]	CDO (H2)[11]	Morsa[2]	EDBM
Set0	Insertion time	-	12.4	19.6	11.8	-	9.2
	Storage size	8.8	29.4	53.6	26	-	36
Set1	Insertion time	-	32.5	57.1	19.2	-	24.6
	Storage size	27	85.9	134	67	-	102
Set2	Insertion time	-	499.1	590.1	778.5	-	247.2
	Storage size	271	794	1197	539	-	643
Set3	Insertion time	-	2210	2245	-	-	647.9
	Storage size	598	1750	2591	-	-	1526
Set4	Insertion time	-	2432	2397	-	-	746.7
	Storage size	646	1890	2789	-	-	1659

and OrientDB and *Morsa_Eval* evaluates XMI, CDO and Morsa. In case of the evaluation of the querying mechanisms, while *NoSQL_Eval* only evaluates the approaches using EOL query language, Morsa_Eval performs evaluations combining approaches with different query languages (Plain EMF, OCL, COCL, EMFQuery, IncQuery and MorsaQL).

5.1 EDBM-Persistence

Two measures have been used for persistence evaluation: (i) time taken to insert each model from XMI to the database (in seconds, s); and (ii) storage size of the models persisted in the databases (in Megabytes, MB). Results are the average duration of the repetitions made for each model insertion (Table 1 describes number of repetitions that have been performed on each case).

Table 2 describes the insertion time and storage size values obtained from EDBM and compares them with the values obtained at *NoSQL_Eval*[11]. *Morsa_Eval*[2] does not contain insertion time and storage size values for Morsa, consequently these values have been omitted.

CDO fails to insert largest models (set3 and set4). Insertion time is similar for all NoSQL based options (Neo4J- and OrientDB-based prototypes) and it rounds 2400 seconds on the largest model (set4). Storage size is around 0.5 times bigger using OrientDB-based prototype than using the Neo4J-based prototype. EDBM performs insertion faster than other approaches, and specially for large models. Insertion times on the largest models (set3 and set4) is around 70% faster than in NoSQL approaches. In terms of storage size, models persisted using EDBM are between 2 and 4 times bigger than the models persisted using XMI but similar to the Neo4J-based prototype.

5.2 EDBM-Query

Query mechanisms have been evaluated using two measurements (executed 100 times each): (i) execution time to return the results of a query (in milliseconds,

		XMI+EMF	XMI+EOL	EDBM
Set0	Time	3419	4462	182
	Mem (max)	155	175	159
	Mem (avg)	122	155	158
Set1	Time	4888	5382	232
	Mem (max)	226	305	159
	Mem (avg)	200	297	156
Set2	Time	20584	26517	1430
	Mem (max)	1161	1138	299
	Mem (avg)	1051	1110	265
Set3	Time	72052	50951	2907
	Mem (max)	2398	2342	318
	Mem (avg)	2297	2289	315
Set4	Time	112406	59722	4021
	Mem (max)	2807	2508	345
	Mem (avg)	2626	2456	322

Table 3: Time (ms) and memory (MB) results obtained from GraBaTs query execution.

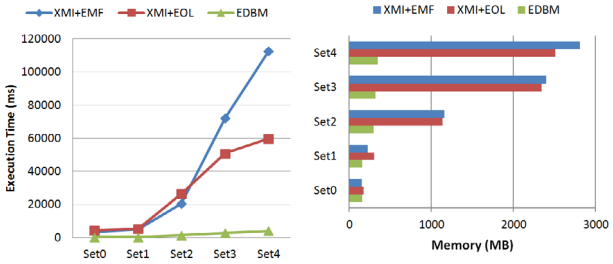
	XMI+EMF	XMI+EOL
Set0	97	74
Set1	99	89
Set2	477	782
Set3	2218	4034
Set4	1090	4339

Table 4: Execution time (ms) to query models that are previously loaded.

ms); and (ii) memory usage (in Megabytes, MB). In the case of EDBM, memory values include: memory used by the embedded database + memory used by the JVM instance. We provide results from: (a) XMI persistence with the query expressed using Plain EMF; (b) XMI persistence with the query expressed using EOL; and (c) EDBM persistence with the query expressed using EOL and runtime query translation.

Execution Time. As is shown in Figure 2a, the size of the model has a great impact over the time required to execute the GraBaTs query if XMI is used. However, results are different depending on whether Plain EMF or EOL has been used to express the query, having greater impact when using Plain EMF (querying the smallest model takes around 3.5 seconds while more than 110 seconds are required for the largest model). XMI+EOL requires around 60 seconds to execute the query over set4, half the time required by XMI+Plain EMF. Table 4 illustrates execution time (in milliseconds) for XMI if the model is previously loaded in memory. In this conditions, XMI+EMF is the fastest option. EDBM, scales better than XMI in terms of execution time: the execution time goes from 182 milliseconds for set0 to 4 seconds for set4. Also, the required execution time grows slower than using XMI. Even comparing with XMI+EMF when models are loaded in memory, the difference is small (3 seconds) and better than XMI+EOL.

Memory Usage. As shown in Figure 2b, memory usage is similar in all three options (155-175 MB) for set0. In case of set1, the maximum memory requirement increases for both XMI-based options (an increase of 71MB for XMI+EMF



(a) Execution time. (b) Memory usage.

Fig. 2: Execution time and memory usage for GraBaTs query in XMI and EDBM

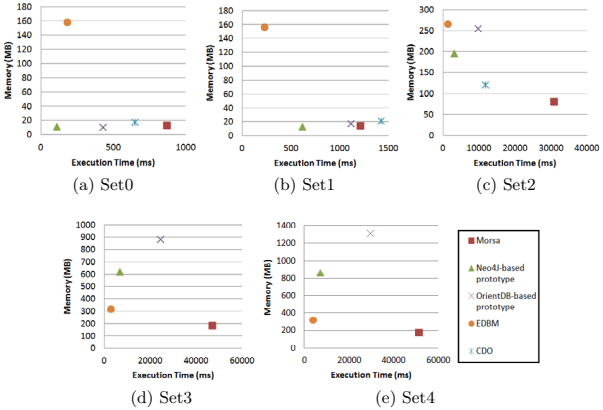
and 130MB for XMI+EOL). The model size has higher impact on memory usage in set2 when XMI is used and it is increased in both cases near 1.1GB. By contrast, memory usage impact is lower if EDBM is used and it increases 140MB. In set3, while memory usage is duplicated respect to set2 when XMI is used (using around 2.3GB), EDBM only requires 318MB. The trend is similar in set4 where XMI+EMF and XMI+EOL both require more than 2.4GB and EDBM only uses 345MB. Memory usage increase is similar on both XMI-based options, they do not scale well in terms of memory. However, EDBM does not require upfront memory loading, and consequently it scales better than XMI (increasing from 159MB on set0 to 345MB on set4).

5.3 EDBM vs. Database Persistence Approaches

Results provided by NoSQL_Eval[11] and Morsa_Eval[2] have been used to compare EDBM performance. As Table 1 describes, different combinations of persistence and query approaches have been evaluated on each study. To facilitate comprehension only the most scalable combinations have been selected: Neo4J-based prototype +EOL and OrientDB-based prototype+EOL from NoSQL_Eval and Morsa+MorsaQL from Morsa_Eval. Since CDO also uses H2 for persistence, we have decided to include it in the comparison. We have used the results of CDO provided by Morsa_Eval.

It is important to note that the execution environment is not the same in all cases. We have not been able to reproduce the evaluation of NoSQL_Eval and Morsa_Eval. Consequently, results of evaluation obtained for such studies are used to compare existing approaches with EDBM. We use these values as a reference to analyse the trend and scalability of our approach.

Figure 3 illustrates time versus memory for each model. We have used the average value of the time measures and the maximum value for the memory usage.



(a) Set0 (b) Set1 (c) Set2

(d) Set3 (e) Set4

Fig. 3: Relation of execution time and memory usage results.

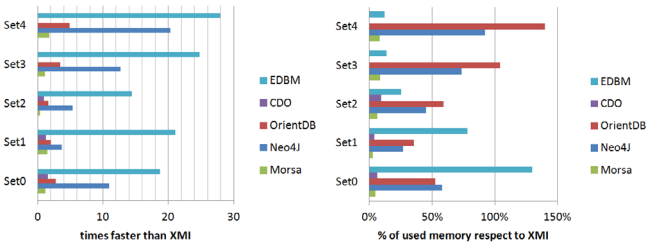
It is important to remark that in EDBM 100 executions have been performed, while 20 and 2 have been made in NoSQL_Eval and Morsa_Eval respectively.

Figure 3a illustrates the time versus memory results for set0. Except EDBM the rest of the approaches use a low amount of memory (between 10-17MB). Neo4J-based prototype is the fastest approach (110ms) having also one of the lowest footprints in memory usage (15MB). In case of Morsa, although memory usage is low (13MB), requires more time than the other options (870ms). In EDBM the memory usage is higher (182MB), but regarding execution time it is placed second (182ms).

Figure 3b illustrates the values for set1. EDBM continues being the approach with the highest memory usage (159MB), but it is also the fastest approach (232ms). Neo4J-based prototype requires the lowest memory amount (18MB) being the second fastest (620ms). CDO is the approach that requires most time (1427ms).

As Figure 3c illustrates set2, Morsa is the approach less memory requirements (81MB), but the slowest (30872ms). EDBM is the fastest one (1430ms) and it is followed by Neo4J-based prototype (3100ms). Although the memory usage continues being higher in EDBM (299MB), in this case, the difference is lower compared to the others.

Results of set3 are illustrated on Figure 3d. Morsa continues being the approach with lowest memory usage value (182MB), but now is followed by EDBM (318MB). EDBM is the fastest one (2907ms), followed by Neo4J-based prototype (6710ms). The OrientDB-based prototype is the approach that uses more



(a) execution time (b) memory usage

Fig. 4: Result comparison in different approaches.

memory (2229MB) and the execution time is 24410ms. CDO values have been omitted since the insertion for set3 failed.

Finally, Figure 3e illustrates the results of set4, very similar to the previous one. In terms of memory the best option is Morsa (180MB), but with a high execution time (51322ms). In terms of time EDBM is the best option with 4021ms and a low memory (345MB).

EDBM is one of the fastest approaches for the five models. However, it is the approach using more memory from set0 to set2. The situation changes in set3 and set4, where it becomes one of the approaches using less memory. These results indicate that EDBM provides an scalable alternative for the persistence and query of large models.

This comparative is more a reference-guide, since execution environments vary. To be able to equally compare, we have standardised the results comparing them with the result of the XMI persistence on each approach. Figure 4 shows: (i) how many times faster is the approach respect to XMI of its related study; and (ii) how much percentage of the memory uses each approach respect to XMI. As Figure 4a illustrates, EDBM and Neo4J-based approach provide more scalability in terms of time to execute the GraBaTs query. Regarding memory, Figure 4b shows that Morsa is the option that scales better. Concerning EDBM, while memory usage is higher in the small models, it scales better as model size increases.

5.4 Threats to Validity

The obtained memory and execution time results, show that our approach is promising in terms of scalability comparing to XMI. Moreover, results indicate that EDBM provides similar scalability to other existing approaches when large models are persisted and queried. However, it has the highest memory usage with smallest models, but still acceptable (e.g. 299MB for set2).

The comparison includes values extracted from different studies, and even if it is useful as reference-guide, executing all the approaches using the same execution environment is more appropriate. Regarding the evaluation, we have selected the GraBaTs 2009 case study, since it is widely used to evaluate the scalability of similar approaches. But using a real industrial domain with real use cases would be more realistic. We plan to perform this task in a future work.

6 Conclusions and Future Work

In this paper, we have evaluated EDBM, an approach for persisting and querying large-scale models. The evaluation is based on the GraBaTs 2009 case study, where large models and a complex query are used to evaluate approaches. Results of the performed evaluations show that (i) EDBM is able to persist large models using a metamodel-agnostic embedded relational database; (ii) runtime translation of EOL queries to SQL, providing a scalable solution to query models persisted in an embedded relational database. We have compared the evaluation results of EDBM, with the results of other existing approaches. These results show that our approach is promising in terms of scalability in contrast to XMI and other persistence approaches.

For future work, we plan to add support for translating model modifications by using EOL. Moreover, we plan to evaluate this approach using an industrial case study. Although EDBM is focused on scalable persistence and query, providing version-control and integration with existing modelling editors is also planed for a future version [23].

Acknowledgments

This work is partially supported by the EC, through the Scalable Modelling and Model Management on the Cloud (MONDO) FP7 STREP project (#611125).

References

1. Benelallam, A., Gómez, A., Sunyé, G., Tisi, M., Launay, D.: Neo4EMF, a Scalable Persistence Layer for EMF Models. In Cabot, J., Rubin, J., eds.: Modelling Foundations and Applications. Volume 8569 of Lecture Notes in Computer Science. Springer International Publishing (2014) 230–241
2. Espinazo Pagán, J., García Molina, J.: Querying Large Models Efficiently. Inf. Softw. Technol. **56**(6) (June 2014) 586–622
3. Carlos, X.D., Sagardui, G., Trujillo, S.: MQT, an Approach for Run-Time Query Translation: From EOL to SQL. In: Proceedings of the 14th International Workshop on OCL and Textual Modelling co-located with MODELS 2014, Valencia, Spain, September 30, 2014. (2014) 13–22
4. Carlos, X.D., Sagardui, G., Murguzur, A., Trujillo, S., Mendialdua, X.: Model Query Translator - A Model-Level Query Approach For Large-Scale Models. In: MODELSWARD 2015 - Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development, Angers, France. (2015)

5. Sottet, J.S., Jouault, F., et al.: Program comprehension. In: Proc. 5th Int. Workshop on Graph-Based Tools. (2009)
6. Pagán, J.E., Cuadrado, J.S., Molina, J.G.: A Repository for Scalable Model Management. Software & Systems Modeling (2013) 1–21
7. Hunt, B.: Mongo-EMF. `https://github.com/BryanHunt/mongo-emf/wiki` (2014) [Online; accessed 30-Jan-2015].
8. Scheidgen, M.: Reference Representation Techniques for Large Models. In: Proceedings of the Workshop on Scalability in Model Driven Engineering. BigMDE '13, New York, NY, USA, ACM (2013) 5:1–5:9
9. Stepper, E.: CDO. `http://eclipse.org/cdo/` (2009) [Online; accessed 30-Jan-2015].
10. Irawan, H., Taal, M.: Teneo/Hibernate. `http://wiki.eclipse.org/Teneo/Hibernate` (2012) [Online; accessed 30-January-2015].
11. Barmpis, K., Kolovos, D.S.: Evaluation of Contemporary Graph Databases for Efficient Persistence of Large-Scale Models. Journal of Object Technology **13**(3) (July 2014) 3:1–26
12. The Mondo Project. `http://www.mondo-project.org/` (2015) [Online; accessed 21-Feb-2015].
13. Shah, S.M., Wei, R., Kolovos, D.S., Rose, L.M., Paige, R.F., Barmpis, K.: A Framework to Benchmark NoSQL Data Stores for Large-Scale Model Persistence. In Dingel, J., Schulte, W., Ramos, I., Abraho, S., Insfran, E., eds.: Model-Driven Engineering Languages and Systems. Volume 8767 of Lecture Notes in Computer Science. Springer International Publishing (2014) 586–601
14. Hunter, A.: Emf query. `https://projects.eclipse.org/projects/modeling.emf.query/` (2015) [Online; accessed 02-February-2015].
15. Ujhelyi, Z., Bergmann, G., Hegedüs, Á., Horváth, Á., Izsó, B., Ráth, I., Szatmári, Z., Varró, D.: EMF-IncQuery: an Integrated Development Environment for Live Model Queries. Sci. Comput. Program. **98** (2015) 80–99
16. Kolovos, D.S., Rose, L.M., Paige, R.F.: The epsilon book (2010)
17. Egea, M., Dania, C., Clavel, M.: MySQL4OCL: A Stored Procedure-Based MySQL Code Generator for OCL. Electronic Communications of the EASST **36** (2010)
18. Heidenreich, F., Wende, C., Demuth, B.: A Framework For Generating Query Language Code From OCL Invariants. Electronic Communications of the EASST **9** (2007)
19. Demuth, B., Hussmann, H., Loecher, S.: OCL as a Specification Language for Business Rules in Database Applications. In Gogolla, M., Kobryn, C., eds.: UML 2001 The Unified Modeling Language. Modeling Languages, Concepts, and Tools. Volume 2185 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2001) 104–117
20. Marder, U., Ritter, N., Steiert, H.: A DBMS-Based Approach for Automatic Checking of OCL Constraints. In: Proceedings of OOPSLA. Volume 99. (1999) 1–5
21. Parreiras, F.S.: Semantic Web and Model-driven Engineering. John Wiley & Sons (2012)
22. Kolovos, D.S., Wei, R., Barmpis, K.: An Approach for Efficient Querying of Large Relational Datasets with OCL-based Languages. In: XM 2013–Extreme Modeling Workshop. (2013) 48
23. Carlos, X.D., Sagardui, G., Trujillo, S.: Scalable Model Edition, Query and Version Control Through Embedded Database Persistence. In: Joint Proceedings of MODELS 2014 Poster Session and the ACM Student Research Competition, Valencia, Spain, September 28 - October 3, 2014. (2014) 11–15

Performance Analysis of Persistence Technologies for Cloud Repositories of Models

Juan Pablo SalazarÁlvarez[1], Elena Gómez Martínez[1], and Miguel de Miguel[1,2]

[1] Center for Open Middlewarre, Universidad Politécnica de Madrid (UPM), Pozuelo de Alarcón - Madrid, Spain
[2] DIT, Universidad Politécnica de Madrid (UPM), Madrid, Spain

Abstract. The growing adoption of Model Driven Development (MDD) in companies during last decade arises some model interchange problems. Companies need support to interchange models and reuse parts of them for developing new projects. Traditional tools for model edition and model interchange have different performance issues related to the models storage. There are mainly two styles to organize the persistence of models into repositories: a complex and large model or a large amount of small models. This last approach is common in companies that generate software from models. In this paper, we analyse performance properties of different persistence technologies to store small/medium-scale models, the analysis results should be considered in the design of model repositories in the cloud. With this aim, we have designed and developed a generic architecture to evaluate each persistence technology under similar situations.

1 Introduction

During the last decade, Model Driven Development (MDD) has been acquired by companies to improve quality, productivity and the reuse of their software development. Models are not only an abstraction of the system under development, but also the system is generated from them. This reuse process is especially important in MDD approaches that generate automatically software code from models. Besides, multinational companies build development teams that are geographically dispersed. MDD developers require virtual model interchange support, which, currently, is well supported for programming languages, but not yet for models. Tools for managing and sharing these models are needed in order to reuse models in a collaborative industrial development ecosystem. Collaboration is usually supported by model interchange and versioning supported in a model repository. Therefore, apart from version control, querying and transformation, one of the key functionality of these tools is the persistence.

In this paper, a model repository in the cloud provides the storage of models with the possibility of accessing and updating such models [3]. The model repository supports any modeling language, even a coexistence of them. To organize models persistence, there are chiefly two styles: either a unique very complex or

large model that contains all the elements or a great number of small models with partial functionality. But also there is a lot in between, a particular example is model fragmentation for large models; in [25], large models can be split and persisted in fragments as small-scale models. There are several works in the literature [3, 4, 10, 11, 14] that evaluate repositories for large-scale models. Nevertheless we focus on the persistence technologies used by those companies that have developed lots of small/medium-scale models. This approach is common in companies that generates software from models. At these companies, the amount of models and their relationships (cross-references) are becoming a problem to be concerned with. Furthermore, maintenance systems are also a weak point in these companies, due to the amount of reused elements among models.

There are two factors that make very complex the construction of applications into a large-scale model: the size of generated software from models (more than 20 times the size of model) and the model accuracy level (because models are the inputs for the construction of executable applications). At this situation, developers tend to decompose their applications into small/medium-scale models (size less than 1M). Santander Group[3] is an example of company that have been using MDD approach for the construction of banking web applications during last decade. Several thousand engineers in different locations use domain specific languages (DSL) for developing software applications. The medium size of these DSL-based models is less than 100K. But each application includes hundreds of models.

There are different alternatives to persist models. By default, modeling tools store models in a standard file approach. They tipically uses XML/XMI for interchange aim. However, other persistence layers have emerged in last years, such as relational databases and NoSQL databases. The relational databases provide mechanisms to index and access objects with SQL queries. The NoSQL databases are based on several *schemaless* database paradigms, among them document or graph.

The ultimate aim of our research is to achieve a framework to share cloud-based small/medium-scale models developed collaboratively whichever modeling language they use. As a first step, we analyse different persistence technologies for performance perspective in this work. For this purpose, we have designed a generic architecture in order to abstract each persistence layer details. The architecture also abstracts the modeling language details. Moreover we have implemented five persistence backends: two file-based solutions and three databases (a relational, a document-based and a graph-based one).

Concerning performance evaluation, we have executed a case study with two usage scenarios to analyse them by means of testing techniques. As performance metrics, we have followed those proposed in Software Performance Engineering (SPE) [26]. Thus, we focus on response time, scalability and storage cost.

The rest of the paper is organized as follows. Firstly, Section 2 outlines the basic concepts and state of the art. Section 3 describes a case study which is analysed in Section 4. Finally, Section 5 states some conclusions.

[3] http://www.santander.com

2 Background and State of the Art

This section defines some concepts related to Model-Driven Development, modeling languages and persistence technologies that will be used in the remainder of the paper. We also outline some collaborative MDD tools.

MDD aims to overcome the complexity of systems and their development by abstracting relevant information, by working at the model instead of the code level. In MDD, models specify the structure, behaviour and requirements of a system. A metamodel provides the syntax specification of a language; it describes the concepts and relationships of a certain domain [10]. Thus, a model is an instance of its corresponding metamodel. Some modeling languages are Business Process Model and Notation (BPMN) [21], the Unified Modeling Language (UML) [20], Meta-Object Facility (MOF) [22] and Ecore [28], among others.

Eclipse Modeling Framework (EMF) [28] is a modeling framework, integrated in Eclipse, for building tools and other applications based on a structured metamodel. EMF facilitates the definition and instantiation of metamodels and runtime support for the models including change notification, persistence implementation and manipulation of EMF objects.

In some sections we use UML and MARTE (Modeling and Analysis of Real-Time and Embedded systems) profile [19] to provide performance information of our benchmarks. MARTE includes the sub-profile Generic Quantitative Analysis Model (GQAM) designed for performance analysis purposes.

2.1 Model Persistence Technologies

There are three chief approaches to store models and metamodels: XML/XMI files, relational databases through object relational mappings and model repositories. In the following, we outline some of them.

Models are typically stored using XML-based formats, like XMI (XML Metadata Interchange) [23], which have to be parsed to build in-memory models. XML parsers are in charge of accessing to its data content and modifying its content or its structure. Parsers carry out two main processes: serialization and deserialization. Serialization process converts an in-memory object into an XML stream. Reversely, deserialization converts XML streams in wire-format objects in memory.

EMF defines the *Resource* interface to represent a physical storage location [28]. The EMF *Resource* provides four basic operations involved in moving models between memory and persistence: load, save, update and delete. *XMLResource* is the interface created by EMF capable for serializing any model into a XML file. In a similar manner, *XMIResource*, inherited from *XMLResource*, is used to serialize any model using XMI 2.0. This is the default serialization adopted by EMF. EMF also provides a simple serialization of the models through *BinaryResource*, which saves the models in binary files.

Teneo [29] is a model-relational mapping and runtime database persistence solution for EMF. Teneo integrates Hibernate and EclipseLink. So, it derives a

relational schema and an object-oriented API from an Ecore metamodel. Relational databases provide mechanisms to index and access objects via SQL queries. Teneo can use MySQL [24], among others, as target database.

Concerning model repositories, Connected Data Objects (CDO) is a *de facto* standard solution to handle models and metamodels in EMF. CDO is both a development-time model repository and a runtime persistence framework [30]. CDO allows models to be persisted into all kinds of database backends like major relational databases or NoSQL databases. However, in practice, only relational databases are commonly used.

Morsa [10,11] is a prototype of a model repository of large scale EMF models based on a non relational database. Morsa uses a document-oriented database as persistence backend, specifically MongoDB [2, 16]. It therefore stores EMF models as collections of documents. The work of Espinazo-Pagán et al. [10, 11] also compares Morsa with CDO and XMI files. As shown, Morsa exhibits better performance than other approaches for large-scale models.

Neo4EMF is a model repository built on the top of the NoSQL graph-based database Neo4j [17]. In Neo4EMF, EMF-based models can be described in terms of graphs concepts, since there is an immediate mapping between the two representations, described in [4]. As the aforementioned model repositories, Neo4EMF aims to scale large scenarios. In [4], a comparison between EMF standards parameters (XMI files), CDO and Neo4EMF is done. The experiments are performed over the Java MoDisco Metamodel [31] and focused on large-scale scenarios. Their results show that Neo4EMF outperforms other alternatives for this kind of scenarios.

EMFStore [14] is a operation-based framework for versioning EMF models, that includes a model repository. Therefore, unlike the above approaches, EMFStore supports model evolutions. Internally, EMFStore handles XMI files that envelopes internal models for operations and changes. EMFStore lacks a mechanism to solve cross-document references between models that have been developed with any other tool.

In this paper, our main objective is the evaluation of persistence mechanisms for large amounts of small-scale models. We have then analysed some of the persistence technologies outlined in this section. Specifically, we have evaluated:

- File-based mechanisms: By means of EMF *Resource*, we have developed two model repositories. The first one uses XMI (*XMIResource*). The second one is a model repository with binary files (*BinaryResource*).
- Relational database-based mechanisms: Since Teneo connector is more simplified than CDO, we have studied Teneo with MySQL database.
- Non-relational databases mechanisms: We have analysed the document-based Morsa and the graph-based Neo4EMF.

We have not evaluated EMFStore, since it was not sufficient mature when implementation phase was carried out.

2.2 Collaborative MDD tools

Insofar our ultimate aim of our research is to achieve an efficient tool for collaborative model-driven development, we briefly outline some of these tools. Although, our aim does not deal with editing, many of these tools also offers a model editor tool among their functionalities.

ModelBus [12] is a tool integration platform for addressing the functional connection of services provided by different modeling tools, such as editing or transformation among others [5]. It is based on a virtual bus-like service-oriented architecture. It allows therefore the data exchange between models in a collaborative environment. As pointed out in [10], the ModelBus repository is a web service application that manages an embedded Subversion engine which implements the actual repository. However, it does not allow partial access to models.

Modelio is an enterprise-level open source modeling solution to develop UML-based models [27]. It supports requests, extraction and modification of models, which can be accessed through a Java API. Modelio uses Teamwork Manager tool to endow collaborative development environment [7]. This module provides a model repository with a central storage area, which is locally replicated by each user. It internally integrates a Subversion repository [1] in order to share modeling projects. Unlike our proposal, EMF is not supported by Modelio [7].

MagicDraw [18] is a commercial modeling framework based on UML standard. By means of Teamwork Server, it provides a central repository for storing models. Models can be accessed, reviewed or modified. Internally, Teamwork Server uses an internal proprietary code to implement a native versioned file-based repository. MagicDraw supports EMF through a conversion into UML.

MetaEdit+ [15] is a commercial domain-specific modeling environment composed of two products: MetaEdit+ Workbench for creating modeling tools and generators and MetaEdit+ Modeler for editing models with multiple users, projects and platforms. It also provides a repository for storing models.

Obeo Designer [9] is a commercial model-driven approach to specify models. It is based on EMF and plugged into Eclipse Platform IDE. Obeo Designer relies on a central CDO repository to facilitate collaborative work [6].

GenMyModel [8] is a commercial modeling platform. It allows to diagram models through web editors among other functionalities, such as to provide a model repository. It focusses on UML, although other languages are available.

3 Case Study: A Generic Architecture

In order to compare the target persistence technologies, a software architecture has been designed for a generic repository. In the following, this architecture and its usage scenarios are described.

3.1 Architectural Design

We have developed a software architecture for a generic repository. The purpose of this architecture is to abstract any modeling language, as well as any persistence solution. The architecture plays with abstract elements to define the same

characteristics and behaviours for different implementations. Thus, the obtained results can be compared without benefit or harm to any technology.

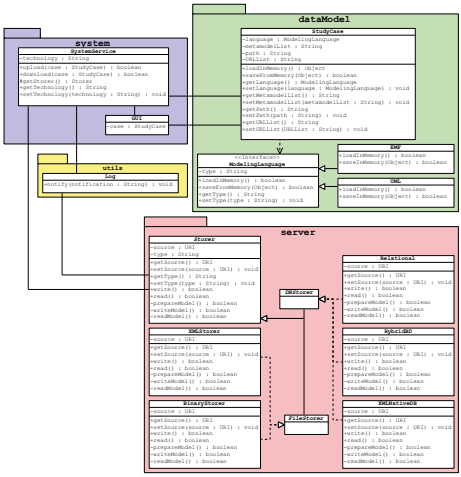


Fig. 1. Class Diagram of the architecture for any repository abstracting persistence.

The generic architecture is depicted in the UML Class Diagram in Figure 1. Its modules are:

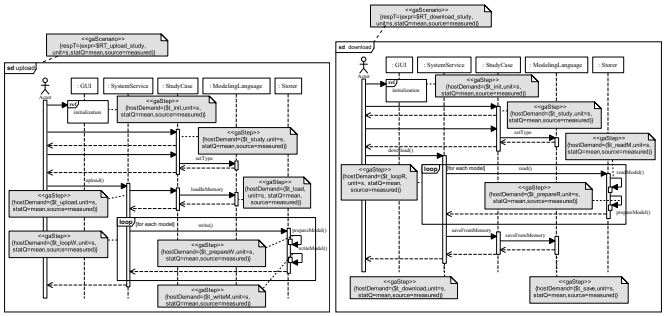
- `system`, which contains the graphical user interface (GUI) and the *SystemService* that can be accessed through the GUI. The technology is also selected in the *SystemService*.
- `dataModel`, whose main class is *StudyCase* which represents the models sets, as well as the modeling language that they support. Thanks to the *interface ModelingLanguage* is possible to unify the behaviour of each language.
- `server`, that contains the *Storer* and their corresponding implementations considered in our analysis, one per each persistence technology. Each storer implements these main functions: *readModel* (recover the model), *writeModel* (persist the model) and *prepareModel* (adapt the model to persistence technology). Eventually, the differences between storers were reduced.
- `utils`, that contains the log for obtaining the metrics.

3.2 Usage Scenarios

According to Smith and Williams [26], performance scenarios are those Use Cases of a software system that are executed frequently or are critical to the user's perception of performance. To compare each technology, we have selected

two main basic usage scenarios of the generic repository as storage location: `Upload` and `Download` scenarios. Firstly, a model developer is able to upload to the generic repository a large number of small-scale models (`Upload`); secondly, a model developer is able to download from the generic repository a large number of small-scale models (`Download`).

Figure 2(a) depicts the UML Sequence Diagram[4] describing the `Upload` scenario. This scenario involves the following steps: i) the user selects the persistence technology and, consequently, ii) the system initializes it taking into account the targeted modeling language; iii) it thus reads the XMI files containing the models, iv) parses them and loads them in memory as EMF *eObjects*; v) it then writes each model to the specific persistence technology in sequence. Previously, each model must be prepared (or processed) according to the specific storer.



(a) `Upload` (b) `Download`

Fig. 2. UML Sequence Diagram describing the `Upload` and `Download` scenarios.

We use the `Download` scenario for the load performance in persistence technologies; the scenario is depicted in the UML Sequence Diagram (SD) of Figure 2(b). This scenario consists of the following steps: i) the initialization of the persistence technology by the user selection; ii) the user chooses the modeling language to recover the stored models. iii) Each model is then read from the specific persistence technology in sequence, iv) recovered and load in memory as an EMF *eObject*; v)then they are copied to XMI files according to the above selected modeling language.

In both scenarios, during step iv) the models are fully load and resolved in order to be able to allow the system to exchange the persistence technology, from the XMI file to the persistence technology selected and vice versa.

As observed, Figures 2(a) and 2(b) are annotated with performance information according to the MARTE profile [19]. These annotations indicate the actions duration collected by experimental tests, as detailed in Section 4.

[4] The reader should note that we have added some grey notes (SD) in the UML diagrams. They are performance annotations that will be briefly outlined in Section 4.

4 Empirical Evaluation

This section compares the selected persistence technologies from software performance viewpoint. For this purpose, the execution environment of the experiments are firstly determined. Then, we define those performance metrics to be evaluated, as well as the experiments results.

4.1 Execution Environment

At this point, we will present results in this section based on two well-known metamodels for benchmarking the selected persistence technologies: Extended Purchase Order (EPO), a metamodel proposed in [28] based on the well-known example from XML Schema Part 0: Primer Second Edition[5]; and Extended Library (LIB), a metamodel taken from an Eclipse tutorial[6]. The two metamodels cover as many features and data types from those available in EMF. In this article we use these metamodels for benchmarking purposes, but we have done similar evaluations and obtained similar results with some other languages such as the *Notation* metamodel that uses Graphical Modeling Framework (GMF) in Eclipse to represent diagrams concepts. We like to emphasize that the metamodels characteristics such as depth of containment tree, number of cross references are irrelevant in the approach of these experiments since the repository is oriented to store models from modeling languages of any kind.

The model instances (models) are automatically generated by Generators which exploits all the attributes and relationships in the metamodels. We have generated two default testbed with 2000 models of EPO and LIB for each one. The number of elements per model are 211 and 313, respectively; the model size is 30 KBytes in EPO metamodel and 47 KBytes in LIB metamodel.

The generic repository has been deployed in a laptop computer running Windows 7 Professional (64 bits) operating system. The computer system characteristics are Intel Core i7 processor 3720QM(2.60GHz) with 16 GB RAM of DDR3 SRAM(1866MHz). Experiments have been executed on Eclipse Helios via Java Application (when it was possible, if not, an Eclipse Application) running Java SE Runtime Environment version 1.7. The Java Virtual Machine (JVM) has been restarted for each measure as well as for each repetition of each measure.

Concerning each specific persistence technology, for all the technologies the default configuration has been used, whenever it was possible and barring error. In particular, we have implemented Teneo/Hibernate with a relational database, MySQL Workbench Community (GPL) for Windows version 6.1.4. As NoSQL databases, we have tested a document-oriented database MongoBD through Morsa [10,11] version 1.0.0 and a graph database named Neo4j via Neo4EMF [4] version 0.1 (November 2013).

[5] http://www.w3.org/TR/smlschema-0/
[6] http://help.eclipse.org/luna/nav/21_1

4.2 Performance Results

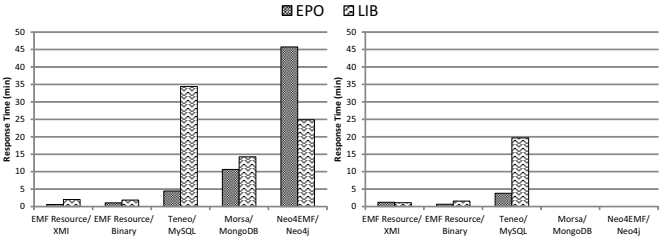
We have compared each persistence technology by considering the following performance properties proposed in [26]: response time, scalability and storage cost. Although there are others, we focus on those critical ones for our aim, a collaborative framework development to share models. In the following, we describe our experiments and the obtained performance metrics in order to analyse them.

Response time *Response time* is the time interval between a user request of a service and the response time of the system [13]. In this paper, the response time can be defined as the scenario duration for each persistence technology.

Performance information concerning atomic actions and scenario durations has been collected in experimental tests. According to MARTE profile, atomic actions are represented by `<<GaStep>>` stereotype, where `hostDemand` tag specifies its corresponding average measured execution time; and scenario durations are specified using `<<GaScenario>>` annotation, as illustrated in Figure 2. Seconds are the measurement unit and mean is the statistical measure.

Figures 3 (a) and (b) illustrate the average response time of `Upload` and `Download` scenarios for each persistence technology. We have analysed the selected persistence technologies considering these situations: i) how it performs depending on the model size; and, ii) how it performs each scenario using the same metamodel. We have obtained these results with the two default testbeds.

As observed in Figures 3 (a) and (b), file-based persistence solutions perform better than database-based ones for both scenarios. Focusing on `Upload` scenario in Figure 3 (a), EMF Resource/XMI outperforms the rest of persistence technologies. Specifically, `Upload` scenario EMF Resource/XMI spends 38.35 seconds and 65.80 seconds for EPO and LIB respectively. As observed, the response time of Teneo/MySQl is the most affected by model sizes, since it is multiplied by 7.5 with LIB models. Remark the values obtained with Neo4EMF/Neo4j, in spite of EPO model size is smaller than LIB one, its `Upload` scenario with LIB models performs better. We guess that this response time is due to Neo4EMF resolves the interrelations within models during the `Upload` scenario.



(a) Upload Scenario (b) Download Scenario

Fig. 3. Response time for each persistence technology by injecting default testbeds.

Concerning `Download` scenario depicted in Figure 3 (b), file-based persistence mechanisms are the fastest solutions. In the case of EMF Resource/Binary, the response time is proportional to the model size. However, response time is not affected by the size in EMF Resource/XMI. As in the previous scenario, Teneo/MySQL performs poorly for LIB models. Furthermore, the response time is almost multiplied by 10 comparing with EPO and LIB models. Note that Morsa/MongoDB has not been performed since prototype could not be run successfully. In the case of Neo4EMF/Neo4j, it was not possible to recover a full load of a model from the repository, so the measures had to be dismissed.

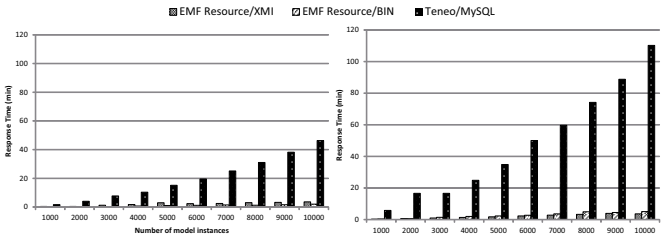
If the same persistence technology are analysed comparing the results in the two scenarios, we observe that `Upload` scenario requires more time than `Download` scenario in file-based mechanisms. An exception is Teneo/MySQL that spends similar response time for both scenarios.

From the response time point of view, we observe that the best persistence solutions (for a very populated model repository of small-scale models) are those based on files. Moreover, it would need to determine what scenarios are the most executed. If we upload more frequently, EMF Resource/XMI performs better than EMF Resource/Binary. If we do not know that information, EMF Resource/Binary behaves more linearly.

Scalability *Scalability* is defined as the ability of a system to continue to meet its response time as the demand for the software function increases [26].

In this paper, the scalability is studied in terms of the time that a testbed needs to perform the `Upload` scenario as the number of models increases. There, we have studied the scalability by varying the number of models injected into each selected persistence technology. Figures 4 (a) and (b) depict the response time when number of models varies from 1000 to 10,000 models of EPO and LIB, respectively.

As can be observed in Figures 4 (a) and 4 (b), EMF Resource/XMI slightly outperforms EMF Resource/Binary files for EPO models. Teneo/MySQL performs significantly poorly compared to file-based mechanisms for both testbeds. Conversely, for LIB models, the response time taken to upload into EMF Re-



(a) EPO (b) LIB

Fig. 4. Response time for `Upload` scenario by varying the number of models injected into each persistence technology.

source/XMI is less than the taken to upload into EMF Resource/Binary. If we analyse how each persistence technology considered in isolation performs, we observe that the response time is linearly with respect its corresponding model size for all solutions. As in previous experiments, Morsa/MongoDB and Neo4EMF/Neo4j have not been included since prototypes could not be run properly for the very populated testbeds. However, these technologies followed the trend set by Teneo/MySQL in the least populated testbeds (for `Upload` scenario).

This set of experiments shows that the best option for large repositories with small-scale models are those based on file, both XMI and EMF Resource/Binary considering the scalability dimension.

Storage cost We define *storage cost* as the amount of disk space used by the system (or technology) in the persistence. The property can also be used to define the space occupied by the system in memory during runtime. Obviously, this property is related to the capacity cost in the case of persistence technologies deployed on a network (cloud-based deployment), since the transmission time of a persisted repository through a network is directly proportional to its size.

To analyse storage cost, we have carried out two set of experiments. The first one compares the storage cost by injecting the two default testbeds into each persistence technology. As second set, we have also studied the impact in the storage cost of increasing the number of models uploaded.

The results obtained in the first set of experiments are shown in Figure 5. They demonstrate that Morsa/MongoDB consumes a lot more disk space than the rest of technologies, specifically two times larger than Neo4EMF/Neo4j and four times larger than Teneo/MySQL. Nevertheless, Morsa/MongoDB uses the same space disk for both metamodels. We guess that it is due to Morsa internally reserves the disk space by blocks. Although it cannot be seen due to the graphic scale, binary files occupy half disk space used by EMF Resource/XMI files.

Figure 6 shows the second set of experiments. It analyses how each persistence technology grows in size when we injected from 1000 to 10,000 models. As can be observed, the storage cost increases linearly in all cases. As aforementioned, two sets of experiments for Morsa/MongoDB and Neo4EMF/Neo4j were not completed, so they have not been included.

Considering the storage cost, the best persistence mechanism for a very populated repository with small-scale models is EMF Resource/Binary, since much less disk space is used.

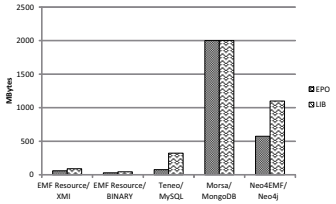
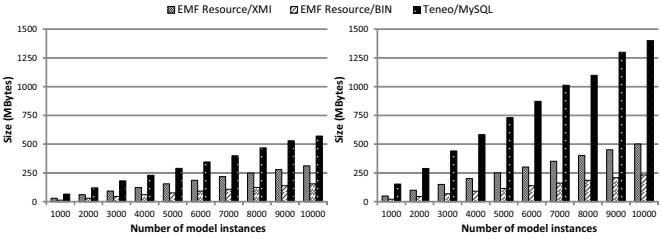


Fig. 5. Storage cost of each persistence technology by injecting default testbeds.



(a) EPO (b) LIB

Fig. 6. Storage Cost by varying the number of models injected into each persistence technology.

4.3 Discussion

The execution of the experiments carried out therefore leads us to determine the best solution for very populated repositories of small/medium-scale models will depend on its use. From the analysis, we have therefore identified the following parameters: i) mean size of metamodels and models, ii) scenarios used frequently; and ii) deployment of the model repository. These three parameters are interrelated, since their variation affects the performance of the others.

Concerning the mean size of metamodels and models, our experiments are focused on small/medium-scale models. As can be observed, file-based solutions outperform database-based ones. Nevertheless, if we compare our results with those obtained in [3, 4, 10] for large-scale models, this kind of file-based mechanisms performs very poorly. Furthermore, since a SAX parser fully reads the XMI file and builds the entire model in memory at once, large models may not be fully kept in memory, causing the parser to overflow the client, as mentioned in [10, 11]. In that cases, a model fragmentation approach as [25] could be considered.

If we take into account what scenario is executed most frequently, response time of `Upload` scenario is slower than `Download`. The former scenario should be scheduled in batch processing at night depending on the persistence solution. For small-scale models, the best solutions are based on XMI and Binary files.

Regarding the deployment of the repository, whether a model repository is deployed onto a network or in the cloud, as a client/server architecture or a web service application, a key aspect is the storage cost of each persistence technology. The time spent to transmit a model to the repository is proportional to its size. As demonstrated, the same models do not consume the same storage cost. In that case, EMF Resource/Binary file as distributed persistence solution is the best option.

We therefore conclude that EMF Resource/Binary file-based mechanism is the best solution for a very populated model repository with small-scale models where the scenario used frequently is `Download` scenario (leaving `Upload` scenario for batch processing).

5 Conclusions and Further Work

Performance constitutes a key aspect of repository-based collaborative modeling to guarantee non-functional requirements, such as response time or scalability. Although there are works in the literature that reporting the comparison of different approaches of model repositories, they only focused on large-scale models. Nevertheless, to the best of our knowledge, we have not identified any work which analyses model repositories to persist large numbers of small-scale models.

In this paper, we explore different persistence technologies and compare from performance perspective. For this purpose, we have designed and implemented a model generic repository to abstract different persistence mechanisms and different modeling languages. This generic architecture allows each persistence technologies to be compared under the same situations.

For future work we have identified a further promising line of research. We aim to implement not only other persistence technologies in our generic architecture, such as XML-native database or EMFStore [14]; but also, other performance issues, such as concurrency or Brewer's Theorem (consistency, availability, partition tolerance). In addition, we have planned to include new functionalities, such as a search engine, a version control system, merge functions and resolution of cross-document dependencies between persisted models. Finally, the obtained results in this paper allow persistence technology to be selected to achieve a collaborative MDD framework for small/medium-scale models.

Acknowledgments The work for this paper was supported by funding from ISBAN and PRODUBAN, under the Center for Open Middleware initiative.

References

1. *Apache Subversion*, 2015. Available at: `https://subversion.apache.org/`.
2. K. Banker. *MongoDB in Action*. Manning Publications Co., 2011.
3. K. Barmpis and D. S. Kolovos. Comparative Analysis of Data Persistence Technologies for Large-scale Models. In *Procs. of the 2012 Extreme Modeling Workshop*, XM '12, pages 33–38, 2012.
4. A. Benelallam, A. Gómez, G. Sunyé, M. Tisi, and D. Launay. Neo4EMF, A Scalable Persistence Layer for EMF Models. In *Modelling Foundations and Applications*, volume 8569 of *LNCS*, pages 230–241. Springer-Verlag, 2014.
5. X. Blanc, M.-P. Gervais, and P. Sriplakich. Model Bus: Towards the Interoperability of Modelling Tools. In *Model Driven Architecture*, volume 3599 of *LNCS*, pages 17–32. Springer Berlin Heidelberg, 2005.
6. H. Brunelière, J. Cabot, S. Drapeau, F. Somda, W. Piers, J. D. Villa Calle, and J.-C. Lafaurie. MDE Support for Enterprise Architecture in an Industrial Context: the TEAP Framework Experience. In *TowArds the Model DrIveN Organization (AMINO 2013) workshop - a MODELS 2013 Satellite Event*, 2013.
7. M. A. Almeida da Silva, A. Abherve, and A. Sadovykh. From the Desktop to the Multi-clouds: The Case of ModelioSaaS. In *15th Int. Symp. on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2013*, pages 462–469. IEEE Computer Society, 2013.

8. M. Dirix, A. Muller, and V. Aranega. Genmymodel: An online uml case tool. *Joint Proceedings of Tools, Demos & Posters*, page 14, 2013.
9. A. El Kouhen, C. Dumoulin, S. Gerard, and P. Boulet. Evaluation of Modeling Tools Adaptation. Technical report, 2012.
10. J. Espinazo Pagán, J. Sánchez Cuadrado, and J. García Molina. Morsa: A Scalable Approach for Persisting and Accessing Large Models. In *Model Driven Engineering Languages and Systems*, volume 6981 of *LNCS*, pages 77–92. Springer-Verlag, 2011.
11. J. Espinazo Pagán, J. Sánchez Cuadrado, and J. García Molina. A repository for scalable model management. *Software & Systems Modeling*, pages 1–21, 2013.
12. Fraunhofer. *ModelBus*, 2015. Available at: `http://www.modelbus.org/`.
13. R. K. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley professional computing. John Wiley, 1991.
14. M. Koegel and J. Helming. EMFStore: A Model Repository for EMF Models. In *Procs. of the 32Nd ACM/IEEE Int. Conf. on Software Engineering - Volume 2*, ICSE '10, pages 307–308. ACM, 2010.
15. MetaCase. *MetaEdit+ Domain-Specific Modeling Environment* . Available at: `https://http://www.metacase.com/products.html`.
16. MongoDB Developers. *MongoDB*. Available at: `http://www.mongodb.org/`.
17. Neo4j Developers. *Neo4j*. Available at: `http://neo4j.com/`.
18. No Magic. *MagicDraw*, 2015. Available at: `http://www.nomagic.com/products/magicdraw.html`.
19. OMG. *A UML profile for Modeling and Analysis of Real Time Embedded Systems (MARTE)*, 2011. Version 1.1 Available at: `http://www.omg.org/spec/MARTE/1.1/`.
20. OMG. *Unified Modeling Language (UML)*, 2012. Version 2.4.1 Available at: `http://www.omg.org/spec/UML/2.4.1/`.
21. OMG. *Business Process Model And Notation (BPMN)*, 2013. Version 2.0.2 Available at: `http://www.omg.org/spec/BPMN/2.0.2/`.
22. OMG. *MetaObject Facility (MOF)*, 2014. Version 2.4.2 Available at: `http://www.omg.org/spec/MOF/2.4.2`.
23. OMG. *XML Metadata Interchange (XMI)*, 2014. Version 2.4.2 Available at: `http://www.omg.org/spec/XMI/2.4.2/`.
24. Oracle. *MySQL*. Available at: `http://www.mysql.com/`.
25. M. Scheidgen, M. Zubow, J. Fischer, and T. H. Kolbe. Automated and transparent model fragmentation for persisting large models. In *Model Driven Engineering Languages and Systems - 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30-October 5, 2012. Proceedings*, pages 102–118, 2012.
26. C. U. Smith and L. G. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison–Wesley, 2002.
27. SOFTEAM. *Modelio*, 2015. Available at: `http://www.modelio.org`.
28. D. Steinberg, F. Budinsky, M. Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.
29. The Eclipse Foundation. *Teneo*. Available at: `http://eclipse.org/modeling/emft/?project=teneo`.
30. The Eclipse Foundation. *The CDO Model Repository*. Available at: `https://eclipse.org/cdo/`.
31. The Eclipse Foundation. *MoDisco Eclipse Project*, 2014. Available at: `https://eclipse.org/MoDisco/`.

Lenguaje específico para el modelado de flujos de trabajo aplicados a ciencia de datos

Rubén Salado-Cid y José Raúl Romero

Dpto. de Informática y Análisis Numérico
Universidad de Córdoba, Campus de Rabanales, 14071 Córdoba
{rsalado,jrromero}@uco.es

Resumen La ciencia de datos permite la extracción de conocimiento utilizando fuentes heterogéneas con grandes volúmenes de datos. Este campo es actualmente una de las áreas de mayor crecimiento debido al aumento exponencial de datos disponibles, con aplicación en un amplio rango de dominios. De hecho, es frecuente que los usuarios no posean conocimientos específicos en computación. Para ellos, los flujos de trabajo son un mecanismo adecuado de representación y modelización de su proceso de trabajo, ya que permiten definir a alto nivel la secuencia de acciones que permitirá capturar la información y transformarla en conocimiento. Los sistemas de gestión de flujos de trabajo son los encargados de ejecutarlos de forma transparente. En este trabajo se presenta, formalizando su sintaxis abstracta, un lenguaje para la definición de flujos de trabajo a distintos niveles de abstracción. El desarrollo de este lenguaje, por su independencia de la notación concreta, se hace necesario para alcanzar un mayor nivel de interoperabilidad entre las aplicaciones ya desarrolladas para este fin.

Keywords: lenguaje específico del dominio, flujos de trabajo, ciencia de datos

1. Introducción

El crecimiento constante de la cantidad de datos disponible ha aumentado el tamaño y la complejidad de la información que necesita ser almacenada y manipulada, provocando que los procesos y técnicas tradicionales de procesamiento de datos hayan dejado de ser eficaces bajo estas condiciones [3]. Esta circunstancia ha dado lugar al crecimiento de áreas de investigación, como la ciencia de datos [6], que tratan de dar solución a los problemas derivados de la ingente cantidad de datos a tratar. La ciencia de datos utiliza conocimientos procedentes de otros campos, como la minería de datos, estadística o aprendizaje automático, para el desarrollo de nuevos procedimientos que permitan la extracción de conocimiento a partir de fuentes de datos heterogéneas, independientemente de las restricciones de almacenamiento, volumen de los datos, capacidad de procesamiento o medios de comunicación. En este contexto, la mayor parte del tiempo de cómputo se emplea en operaciones de transporte y procesamiento de grandes cantidades de datos, lo que es denominado como *computación intensiva en datos*.

La computación intensiva en datos [5] utiliza los principios de la paralelización de los datos y de la computación distribuida para procesar enormes conjuntos de datos. Este tipo de computación es escalable con respecto a los recursos computacionales empleados, proporciona alta disponibilidad, tolerancia a fallos y alto rendimiento debido al uso de numerosos nodos de procesamiento con una alta tasa de acceso a disco, y es flexible a la hora de integrar otro tipo de tecnologías de procesamiento de datos. Actualmente, algunas de las técnicas y arquitecturas para la computación intensiva en datos [2,17] ya están siendo ampliamente explotadas para el desarrollo de aplicaciones para ciencia de datos.

Sin embargo, existe una gran dificultad en el uso de las técnicas de ciencia de datos por parte de muchos de los usuarios que las requieren en sus dominios de trabajo, como expertos en astronomía, medicina o biología, debido a la necesidad de poseer profundos conocimientos en áreas relacionadas con la informática como la minería de datos, computación paralela, computación distribuida o el desarrollo avanzado de software. Por este motivo se hace necesario proporcionar mecanismos que faciliten al experto, que no posee estos conocimientos avanzados en computación, la representación y definición de la secuencia de acciones necesaria para capturar la información de su dominio y convertirla en conocimiento sin tener que especificar los detalles de bajo nivel de ejecución.

El uso de flujos de trabajo [1] para la especificación de procesos de análisis de información y extracción de conocimiento proporciona un nivel de abstracción adecuado para este tipo de usuarios. Un flujo de trabajo permite capturar y modelar el conocimiento del experto como una secuencia de acciones o tareas que trabajan en coordinación para llevar a cabo un objetivo determinado, favoreciendo además la automatización de su ejecución por parte de los denominados *sistemas de gestión de flujos de trabajo* [18]. De esta forma, los aspectos de bajo nivel de computación son transparentes al usuario en este tipo de sistemas, incrementando su productividad y reduciendo el *time-to-value*.

Actualmente ya existen sistemas de gestión de flujos de trabajo en distintos campos de aplicación intensivos en datos, como bioinformática [11], minería de datos [8] o metaheurísticas [13]. Sin embargo, ninguno de estos sistemas formaliza la sintaxis abstracta del lenguaje con el que definen los flujos de trabajo, lo que conlleva la incompatibilidad e imposibilidad de reutilizar el conocimiento capturado de un sistema a otro. Existe la necesidad de estandarizar la definición de flujos de trabajo en dominios intensivos en datos para el desarrollo de herramientas interoperables, tal y como se indica en [14].

En este trabajo se presenta la formalización de la sintaxis abstracta de un lenguaje específico de dominio para el modelado de flujos de trabajo aplicados a ciencia de datos. Este lenguaje define un conjunto de elementos, relaciones y restricciones que permiten expresar a un alto nivel de abstracción la secuencia de tareas a ejecutar para realizar el proceso de análisis de la información y extracción de conocimiento en cualquier dominio de trabajo. De esta manera, también se favorece la posterior utilización de técnicas de transformación de modelos para alcanzar la interoperabilidad entre diferentes sistemas de gestión de flujos de trabajo. Además, para mostrar la aplicabilidad del lenguaje, se

introduce un ejemplo de uso donde se define un flujo de trabajo para el análisis de información en un dominio científico intensivo en datos, utilizando una sintaxis concreta definida a modo ilustrativo.

En el resto del artículo se presenta en profundidad el lenguaje específico de dominio propuesto. En la Sección 2 se introducen los antecedentes de este trabajo. En la Sección 3 se describe la formalización de la sintaxis abstracta del lenguaje. En la Sección 4 se muestra un ejemplo de uso donde se define un flujo de trabajo aplicado a ciencia de datos. Finalmente, en la Sección 5 se exponen las conclusiones obtenidas, destacando algunas líneas de trabajo futuro.

2. Antecedentes

Tradicionalmente, los flujos de trabajo han sido utilizados en otros ámbitos, como los procesos de negocio. La gestión de procesos de negocio [16] es un área relacionada con la especificación y mejora de los procesos existentes dentro de un entorno industrial, que utilizan distintos tipos de tecnologías para facilitar la coordinación y comunicación entre los diferentes agentes involucrados, pudiendo ser tanto agentes humanos como informatizados. WS-BPEL [9] es el lenguaje estandarizado por OASIS para la definición y ejecución de flujos de trabajo para la gestión de procesos de negocio como servicios web.

Sin embargo, la gestión de procesos de negocio tiene objetivos distintos a la computación intensiva en datos [7]. En la gestión de procesos de negocio, el foco recae sobre la orquestación de los recursos humanos y de los sistemas de información involucrados para alcanzar un objetivo determinado, mientras que la computación intensiva en datos se centra en la ejecución eficiente de una serie de actividades computacionales, optimizando todos los recursos disponibles e integrando distintos tipos de datos. Por tanto, estas diferencias provocan que no resulte adecuado reutilizar estos lenguajes en dominios diferentes para los que fueron desarrollados.

Para el caso de la definición de procesos basados en flujos de trabajo en áreas de computación intensiva de datos, como la ciencia de datos, no existe actualmente ningún lenguaje estandarizado para su especificación. Sin embargo, existen herramientas que internamente utilizan un lenguaje propio para la especificación de este tipo de procedimientos.

Taverna [11] es un sistema de gestión de flujos de trabajo que permite la definición, ejecución e intercambio de flujos de trabajo científicos. Proporciona acceso a numerosos servicios procedentes de distintas áreas, como bioinformática, astronomía o informática química, además de permitir la invocación de servicios genéricos que dispongan de un documento WSDL [15]. Por defecto, el orden de ejecución de los componentes que conforman los flujos de trabajo es derivado de las dependencias de datos existentes entre las distintas operaciones a ejecutar, en vez de ser definido por el usuario. No obstante, es posible establecer un orden de ejecución secuencial de forma explícita.

KNIME [8] es una plataforma de minería de datos que utiliza los flujos de trabajo como método para la modelar y ejecutar procedimientos de análisis de

datos. Proporciona numerosos tipos de algoritmos para la manipulación, visualización, análisis, y entrada y salida de datos, que principalmente son implementados utilizando Java. Sin embargo, es posible incorporar nuevos algoritmos desarrollados con lenguajes de programación como Java, R o Python. La ejecución de los flujos de trabajo es llevaba a cabo por un motor de ejecución que deriva el orden de los componentes del flujo de trabajo únicamente de las dependencias de datos existentes entre ellos.

También existen sistemas que aplican los flujos de trabajo sobre otro tipo de dominios, como el prototipo presentado en [13]. Esta herramienta proporciona un entorno gráfico para la definición y ejecución visual de flujos de trabajo para la computación evolutiva. Proporciona los elementos funcionales necesarios para la especificación de algoritmos evolutivos, como operadores de mutación, cruce y selección, codificación de individuos, etc. Además, facilita la creación de un marco de experimentación para validar los resultados obtenidos por los flujos de trabajo compuestos anteriormente, incluyendo distintos tests estadísticos. La funcionalidad de la herramienta puede ser extendida incluyendo nuevos algoritmos desarrollados en Java. El orden de ejecución de los componentes del flujo de trabajo es explícitamente establecido por el usuario, mediante líneas de conexión que establecen dependencias de control entre ellos.

Por tanto, es necesario formalizar la sintaxis abstracta de un lenguaje para la especificación de flujos de trabajo para ciencia de datos, de forma que se favorezca la extracción de conocimiento en todos aquellos dominios de trabajo que lo necesiten. Este lenguaje debe reunir las principales características de los sistemas que ya están utilizando flujos de trabajo para este propósito, con el fin de que pueda ser utilizado como base para alcanzar la interoperabilidad entre sistemas y para el desarrollo de nuevas herramientas en todo tipo de dominios intensivos en datos.

3. Sintaxis abstracta del lenguaje

Actualmente, ninguno de los lenguajes de flujos de trabajo existentes para computación intensiva en datos formaliza su sintaxis abstracta, lo que obliga a deducir un metamodelo a partir de una especificación informal. La sintaxis abstracta del lenguaje propuesto contiene las principales características y elementos comunes de estos lenguajes, lo que permite la interoperabilidad entre los distintos sistemas de gestión de flujos de trabajo. Además, se han incorporado nuevos elementos para satisfacer requisitos específicos de la ciencia de datos, como distintos proveedores de datos o metainformación sobre el contexto del flujo de trabajo.

La estructura del metamodelo propuesto se ha separado en capas para facilitar la comprensión, agrupando los elementos con funcionalidad similar. En la Figura 1 se ilustra dicha organización:

Capa estructural. Contiene todos los elementos necesarios para estructurar los flujos de trabajo en forma de grafo dirigido, donde los vértices representan los diferentes tipos de unidades funcionales a ejecutar y los arcos indican las distintas dependencias existentes.

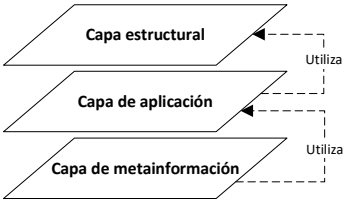


Figura 1. División por capas del lenguaje

Capa de aplicación. Agrupa los elementos que permiten capturar los requisitos específicos del dominio, como las tareas a ejecutar, los tipos de datos soportados, la definición de los recursos computacionales mínimos o la gestión de los errores de ejecución.

Capa de metainformación. Contiene los elementos que permiten asociar información acerca del contexto en el que ha sido desarrollado el flujo de trabajo o sobre los autores del mismo. Esta información está destinada a ser consumida por los propios usuarios, no a ser interpretada durante la ejecución del flujo de trabajo.

Obsérvese que no todos los elementos tienen que estar presentes en la definición de un flujo de trabajo con este lenguaje. Esto se debe a las particularidades de cada dominio y, también, a que algunos de los elementos podrían ser generados automáticamente por un sistema de gestión de flujos de trabajo o herramienta similar, por lo que podría omitirse su uso por el experto. El lenguaje puede ser extendido para adaptarlo a especificaciones concretas, ampliando la definición de determinadas metaclases, estereotipadas como «*extension point*».

3.1. Capa estructural

En la Figura 2 se muestran todos los elementos del lenguaje que, junto con sus relaciones, conforman esta capa. El lenguaje define un flujo de trabajo como un grafo dirigido (*ExecutionGraph*) que contiene un conjunto de nodos de ejecución (*Node*) y conexiones que expresan dependencias entre ellos (*DirectedEdge*). Esta estructura en forma de grafo se adecua a la especificación del proceso a modelar como una serie ordenada de acciones y favorece su comprensión por los usuarios.

Un nodo es un elemento con un nombre (*NamedElement*) que representa una unidad básica de ejecución y contiene un conjunto de puntos de conexión (*Endpoint*). Éstos le permiten interaccionar con el resto de nodos. Cada nodo admite distintos tipos de conexiones según los puntos de conexión que contenga, los cuales pueden ser para aceptar conexiones de control (*ControlEndpoint*), conexiones de datos (*DataEndpoint*) o conexiones de excepción (*ExceptionEndpoint*).

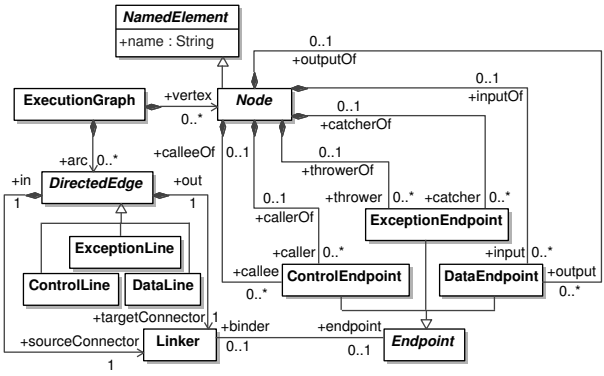


Figura 2. Metamodelo de la capa estructural

Una conexión permite establecer una relación de dependencia entre dos nodos distintos. Dependiendo del tipo de relación, una conexión puede expresar una dependencia de control (*ControlLine*) si la conexión determina el orden secuencial de ejecución entre ambos nodos, una dependencia de datos (*DataLine*) si especifica un intercambio de información entre los nodos, o una situación de excepción (*ExceptionLine*) si determina el nodo alternativo de ejecución en caso de que un error sea detectado. Un elemento vinculante (*Linker*) se encarga de relacionar cada nodo a través de los puntos de conexión, especificando tanto el nodo origen como el destino.

3.2. Capa de aplicación

En la Figura 3 se muestra la vista parcial de los principales elementos del metamodelo de esta capa de aplicación. Algunos elementos del metamodelo se omiten por motivos de espacio. Todo flujo de trabajo (*Workflow*) definido con este lenguaje está compuesto por una o varias unidades funcionales, denominadas *constructores* (*Constructor*), que permiten llevar a cabo un tipo de tarea específico. Un flujo de trabajo puede tener un orden de ejecución explícitamente definido por el usuario, o bien pueden ser las dependencias existentes entre los datos las que determinen el control de ejecución. También, un flujo de trabajo puede permitir que el orden de ejecución sea tanto explícitamente definido como derivado de las dependencias de los datos. Cada tipo de flujo de trabajo determina los tipos de constructores y conexiones admitidos.

Dependiendo de la funcionalidad proporcionada, un constructor puede ser definido como actividad (*Activity*), proveedor de datos (*DataProvider*) o estructura de control (*ControlStructure*). Una actividad representa la transformación

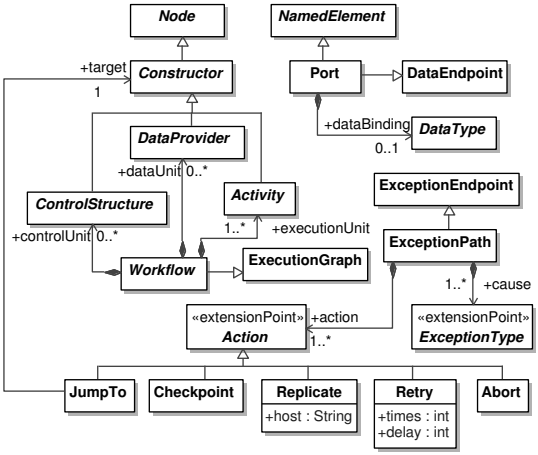


Figura 3. Metamodelo de la capa de aplicación

de un conjunto de datos de entrada para generar datos de salida. Las actividades pueden ser especializadas (Figura 4) como tareas computacionales (*ComputationalTask*) o de visualización (*DisplayTask*). Las tareas computacionales pueden ser ejecutadas sin necesidad de interacción por parte del usuario, tratándose tanto de procesos (*Process*) como de servicios (*Service*). Los tipo de procesos soportados son procesos Java (*JavaProcess*) para la ejecución de código escrito en Java, procesos CLI (*CLIProcess*) para ejecutar programas que utilicen una interfaz por línea de comandos, y flujos de trabajo previamente definidos con el propio lenguaje. No obstante, éstos pueden ser extendidos para referirse a otros lenguajes, plataformas y tecnologías.

También son soportados distintos tipo de servicios web (*WebServices*), como servicios SOAP (*SOAPClient*) y servicios REST (*RESTClient*). Al igual que los procesos, este tipo de servicios es extensible en el lenguaje. Además, las tareas computacionales permiten la definición de los recursos computacionales requeridos para su ejecución, como las características de la máquina específica donde ejecutar, el tipo de sistema operativo compatible o las características mínimas requeridas del procesador. Esta definición es compatible y puede ser extendida con el lenguaje estandarizado por OGF (Open Grid Forum) para este propósito [10]. Por otro lado, las tareas de visualización determinan cómo se deben mostrar gráficamente los datos gestionados en el flujo de trabajo. Este tipo de tarea puede establecer que la visualización de los datos sea delegada en una aplicación externa (*DelegatedDisplay*), o que sea integrada dentro de la propia aplicación que gestiona el flujo de trabajo (*EmbeddedDisplay*).

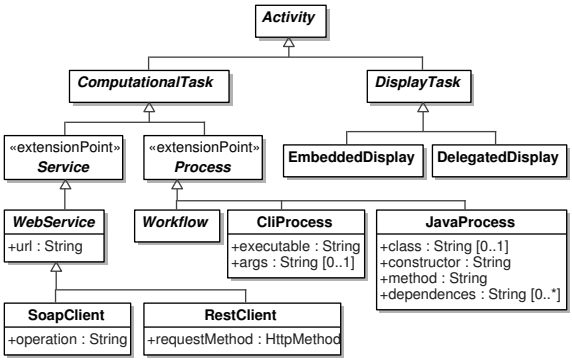


Figura 4. Vista parcial de las actividades de la capa de aplicación

Una estructura de control permite controlar explícitamente el orden de ejecución de los distintos elementos que componen el flujo de trabajo. Para ello, se proporcionan distintos tipos de estructuras de control que permiten indicar el punto de comienzo y de finalización del flujo de trabajo, establecer el siguiente constructor a ejecutar en base al cumplimiento, o no, de una expresión lógica, dividir el flujo de ejecución en dos o más flujos de ejecución paralelos que serán ejecutados de forma concurrente, unificar y sincronizar flujos de ejecución paralelos y, finalmente, contar el número de veces que el flujo de ejecución pasa a través de un elemento específico, lo que permite la realización explícita de bucles.

Por su parte, un proveedor de datos (Figura 5) define el lugar de origen y de destino de los datos que van a ser manipulados dentro del flujo de trabajo. Dependiendo del lugar donde los datos se encuentran almacenados, un proveedor de datos puede ser un registro (*Record*) que permite leer y escribir información en un bloque de memoria principal, un fichero (*File*) o una base de datos (*Database*).

Cada uno de estos constructores contiene una serie de puertos (*Port*), que permiten el envío y recepción de información entre ellos. Un puerto es identificado por un nombre y puede ser asociado con un tipo de dato (*DataType*) específico, limitando el tipo de información aceptada. El lenguaje proporciona una serie de tipos de datos básicos, que se corresponden con los tipos de datos primitivos de cualquier lenguaje de programación tradicional (valores enteros, booleanos o caracteres), y tipos de datos complejos, que se corresponden con estructuras de datos más complejas (vídeos, audios, imágenes o documentos XML).

Además, también son definidos elementos para la gestión de situaciones alternativas (*ExceptionPath*) cuando se producen errores inesperados en tiempo de ejecución (*ExceptionType*). Los tipos de errores soportados por el lenguaje permiten detectar si un tipo de dato es incompatible, si la memoria secundaria

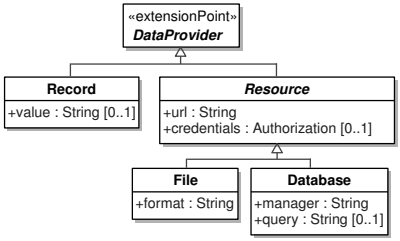


Figura 5. Vista parcial de los proveedores de datos de la capa de aplicación

está completa, si el permiso de acceso es denegado a un determinado recurso, si el recurso no es accesible temporalmente, si se produce un error genérico en tiempo de ejecución, si el tiempo de espera ha finalizado o si se produce un error de origen desconocido. Las acciones (*Action*) que se pueden establecer para gestionar dichas situaciones permiten continuar y ejecutar un constructor determinado (*JumpTo*), guardar el estado actual de la ejecución para continuar cuando el sistema se recupere (*Checkpoint*), intentar la ejecución en otro lugar indicando su dirección (*Replicate*), detener la ejecución y finalizar (*Abort*) o reintentar la ejecución un determinado número de veces máximo con un tiempo de espera específico entre cada intento (*Retry*). Tantos los tipos de excepción como las acciones de resolución de conflictos son extensibles en el lenguaje.

3.3. Capa de metainformación

En la Figura 6 se muestran todos los elementos que conforman esta capa del lenguaje. Cualquier flujo de trabajo tiene asociado una descripción (*WFSpecification*) que permite definir diversos aspectos relacionados con el contexto que ha propiciado su desarrollo. Este tipo de información no tiene que ser tenida en cuenta por los sistemas de gestión de flujos de trabajo, sino que está destinada a ser utilizada como documentación para el usuario.

Los elementos definidos permiten indicar información acerca del proyecto (*Project*) que ha impulsado la especificación del flujo de trabajo. Un experimento (*Experiment*) es un tipo de proyecto que se refiere a un procedimiento de estudio científico. Actualmente, se permite definir un experimento básico (*BasicExperiment*), que contiene la hipótesis científica (*Hypothesis*) a confirmar, y un conjunto de parámetros para definir la configuración y restricciones (*Configuration*) que deben ser consideradas durante su realización. La definición de un experimento científico es compatible y puede ser ampliada con el lenguaje SEDL presentado para este propósito en [12]. A su vez, un proyecto está compuesto por uno o varios participantes (*Stakeholder*), que puede ser tanto una organiza-

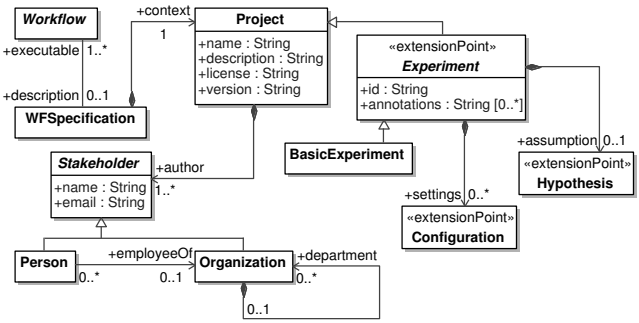


Figura 6. Metamodelo de la capa de metainformación

ción (*Organization*) compuesta por departamentos, como una persona (*Person*) asociada a una organización.

4. Ejemplo de uso

Para mostrar la aplicabilidad del lenguaje propuesto, se ha realizado un ejemplo de uso en el que se define un flujo de trabajo aplicado a ciencia de datos. Debido a que su principal objetivo es facilitar la especificación de procesos de extracción de conocimiento como una secuencia de tareas a ejecutar por parte de usuarios no expertos en computación, se ha propuesto una sintaxis concreta, a modo de ejemplo, que permite representar los constructores del lenguaje con una serie de elementos gráficos. No obstante, el lenguaje es independiente de cualquier sintaxis concreta y permite su representación tanto de forma gráfica como textual.

4.1. Especificación de la sintaxis concreta

La sintaxis concreta implementada a modo de ejemplo asocia una serie de figuras gráficas básicas a los principales constructores del lenguaje. Esta sintaxis está inspirada en la propuesta por UML para la representación de los diagramas de actividades, puesto que algunos de los constructores del lenguaje presentan una funcionalidad similar. En la Figura 7 se muestran los principales constructores con la representación gráfica asociada. Por simplicidad, el constructor *Activity* es representado utilizando la misma figura para cada una de sus distintas especializaciones. Los constructores *Begin*, *End*, *Conditional*, *Merge*, *Fork*, *Synchronizer* y *Counter* son especializaciones de la metaclase *ControlStructure*, omitidos del metamodelo por motivos de espacio. Esta sintaxis será utilizada

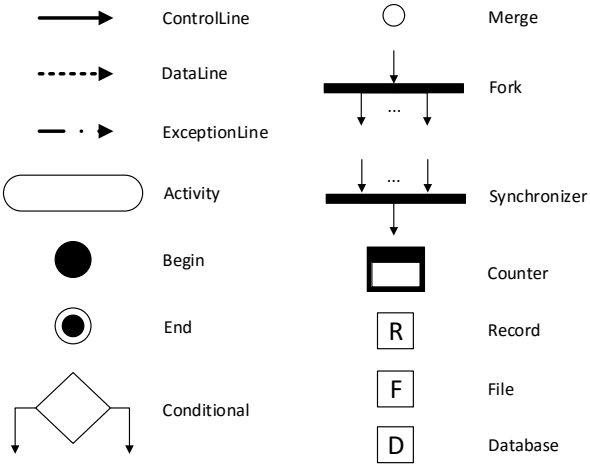


Figura 7. Representación gráfica parcial del lenguaje

para la definición de un flujo de trabajo para ciencia de datos explicado a continuación. Obsérvese que junto al elemento gráfico se ha indicado el nombre de la metaclase del lenguaje que representa.

4.2. Definición de un flujo de trabajo para ciencia de datos

La Figura 8 muestra un flujo para el análisis y extracción de conocimiento en el dominio intensivo en datos de la bioinformática, obtenido de un repositorio público de flujos de trabajo científicos [4]. Para su representación se ha utilizado la sintaxis concreta de la sección 4.1. Debido a que se trata de un flujo de trabajo dirigido por los datos, el lenguaje admite solamente la inclusión de constructores de tipo *Activity* y de tipo *DataProvider*, además de las líneas de tipo *DataLine* y *ExceptionLine*.

El objetivo del flujo de trabajo definido es encontrar posibles enfermedades en base a una serie de términos introducidos por el usuario. Para ello se realiza una consulta, a través de un servicio SOAP, a una base de datos de bibliografía médica a través de la actividad correspondiente (*Retrieve documents*). Esta actividad recibe como parámetros de entrada el número máximo de documentos que pueden obtenerse como respuesta a dicha consulta (*Max number of docs to retrieve*), una cadena con los términos que deben contener los documentos que se

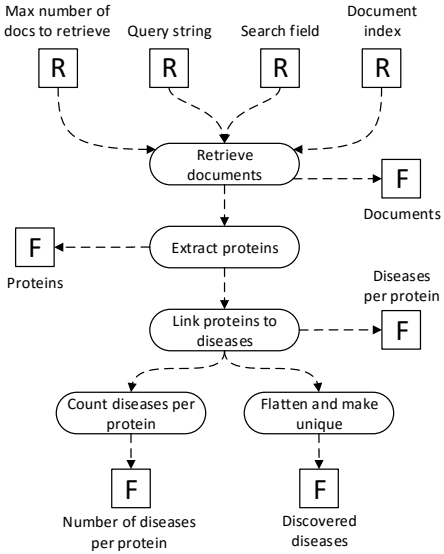


Figura 8. Ejemplo de flujo de trabajo para bioinformática

desean consultar (*Query string*), la parte del documento donde se van a buscar dichos términos (*Search field*) como, por ejemplo, en el título o en el contenido, y el nombre de la base de datos donde se va a realizar la búsqueda (*Document index*) como, por ejemplo, MedLine.

Una vez realizada esta consulta, se obtiene una lista con todos los documentos que cumplen los requisitos especificados. Dicha lista es almacenada en un fichero (*Documents*) y es utilizada por la actividad (*Extract proteins*) para el descubrimiento de proteínas. Esta actividad ejecutará un código escrito en Java para obtener una lista con todas las proteínas citadas en los documentos. Las proteínas descubiertas son almacenadas en un fichero (*Proteins*) y son manipuladas por una actividad (*Link proteins to diseases*) que permite asociar dichas proteínas a determinadas enfermedades. Esta asociación se hace mediante una invocación a un servicio SOAP, que accede a una base de datos de enfermedades. El resultado de esta invocación es una lista que relaciona las enfermedades posibles con cada proteína, la cual es almacenada en un fichero (*Diseases per protein*) y sirve como parámetro de entrada a las dos actividades siguientes.

Por un lado, una actividad (*Count diseases per protein*) utiliza un código escrito en Java para contar el número de enfermedades posibles asociadas a cada proteína. El resultado de esta actividad es almacenada en un fichero (*Number of diseases per protein*). Por otro lado, una actividad (*Flatten and make unique*) manipula la lista de entrada para generar un documento en un formato legible por los humanos con las enfermedades descubiertas. Este documento es también almacenado en otro fichero (*Discovered diseases*). Como se puede comprobar, el modelado del proceso de análisis de información permite la ocultación de los detalles de bajo nivel de ejecución. Este proceso es definido como un conjunto de tareas que realizan distintos tipos de procesamiento sobre grandes volúmenes de datos para extraer conocimiento útil para el experto.

5. Conclusiones y trabajo futuro

En este artículo se ha presentado la formalización de la sintaxis abstracta de un lenguaje para el modelado de flujos de trabajo que permiten la definición de procedimientos de análisis de información y extracción de conocimiento. Actualmente no existe ningún lenguaje de este tipo cuya sintaxis abstracta haya sido formalizada para este propósito. Los sistemas actuales que utilizan flujos de trabajo sobre este tipo de dominios intensivos en datos suelen definir su propio lenguaje de forma interna, con sus propias características, lo que provoca que los procedimientos desarrollados no puedan ser reutilizados en otros sistemas de similares características, ni que éstos puedan ser compatibles entre sí. Todo esto dificulta la labor del usuario, al estar limitada la interoperabilidad de sus flujos de trabajo por el lenguaje que implementa el sistema utilizado.

El lenguaje propuesto sirve como base para el desarrollo de nuevos sistemas basados en flujos de trabajo al proporcionar una formalización de los conceptos, relaciones y restricciones conforme a las reglas del dominio. Esta formalización permite lograr la interoperabilidad entre los distintos sistemas de gestión de flujos de trabajo mediante la transformación de modelos. No obstante, obsérvese que la sintaxis abstracta de estos lenguajes habrá de extraerse directamente de la representación concreta de los flujos de trabajo definidos ya que ninguno de los sistemas actualmente disponibles formaliza su sintaxis abstracta.

Además, para mostrar la aplicabilidad del lenguaje se ha presentado un ejemplo de uso. Primero se ha desarrollado una sintaxis concreta a modo ilustrativo. Luego, se ha definido un flujo de trabajo para bioinformática que permite descubrir enfermedades a partir de una serie de términos específicos. Para lograr este propósito, se especifican los distintos servicios y procesos que serán utilizados, estableciendo las dependencias de datos que existen entre ellos.

La línea a seguir en el trabajo futuro consiste en incluir nuevos tipos de constructores que amplíen la variedad de procesos y servicios soportados, como procesos desarrollados usando lenguajes de programación como R o Ruby, y el desarrollo de un catálogo de transformaciones de modelos que permitan lograr la interoperabilidad entre los distintos sistemas actuales de gestión de flujos de trabajo, como Taverna o KNIME.

Agradecimientos

Trabajo apoyado por el Ministerio de Economía y Competitividad, proyecto TIN2014-55252-P.

Referencias

1. Terminology & glossary. Tech. Rep. WFMC-TC-1011, Workflow Management Coalition (1999)
2. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
3. Fan, W., Bifet, A.: Mining Big Data: Current status, and forecast to the future. SIGKDD Explorations Newsletter 14(2), 1–5 (2013)
4. Goble, C.A., De Roure, D.C.: myExperiment: Social networking for workflow-using e-scientists (2007)
5. Gorton, I., Greenfield, P., Szalay, A., Williams, R.: Data-intensive computing in the 21st century. Computer 41(4), 30–32 (2008)
6. Loukides, M.: What is data science? O'Reilly radar (2010)
7. Ludäscher, B., Weske, M., McPhillips, T., Bowers, S.: Scientific workflows: Business as usual? Business Process Management pp. 31–47 (2009)
8. Mazanetz, M., Marmon, R., Reisser, C., Morao, I.: Drug discovery applications for KNIME: An open source data mining platform. Current Topics in Medicinal Chemistry 12(18), 1965–1978 (2012)
9. OASIS: Web services business process execution language version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf` (2007)
10. OGF: Job submission description language (JSDL) specification, version 1.0. `http://www.ogf.org/documents/GFD.136.pdf` (2008)
11. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M., Wipat, A., Li, P.: Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20(17), 3045–3054 (2004)
12. Parejo, J.A.: MOSES: A metaheuristic optimization software ecosystem. Ph.D. thesis, University of Sevilla (2013)
13. Salado-Cid, R., Luque, G., Romero, J.R.: Sistema de gestión de flujos de trabajo para la definición visual de aplicaciones basadas en algoritmos evolutivos. XVI Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA) (2015)
14. Salado-Cid, R., Romero, J.R., Ventura, S.: Metaherramienta para la generación de aplicaciones científicas basadas en workflows. X Jornadas de Ciencia e Ingeniería de Servicios (JCIS) pp. 96–105 (2014)
15. W3C: Web services description language version 1.1. `https://www.w3.org/TR/wsdl` (2001)
16. Weske, M.: Business process management: Concepts, languages, architectures. Springer Berlin Heidelberg (2012)
17. White, T.: Hadoop: The definitive guide. O'Reilly, first edn. (2009)
18. Yu, J., Buyya, R.: A taxonomy of workflow management systems for grid computing. Journal of Grid Computing 3(3), 171–200 (2006)

MDDE: Una concepción genérica para diseño de entornos de desarrollo de software basados en MDSE

César Cuevas, Patricia López Martínez y José M. Drake

Grupo de Ingeniería Software y Tiempo Real, Universidad de Cantabria

{cuevasce,lopezpa,drakej}@unican.es

Resumen. Se presenta ***MDDE*** (*Model-Driven Development Environment*), una concepción genérica para diseño de entornos de desarrollo de software basados en MDSE. Su objetivo es facilitar el uso de esta disciplina a los ingenieros software que diseñan e implementan entornos de soporte a las metodologías que proponen y que necesitan incluir en ellos nuevos modelos de información, herramientas y procesos de desarrollo. El componente principal de la concepción propuesta es un modelo de referencia que define las capacidades básicas, tanto funcionales como de interacción, que son comunes a cualquier entorno. En ella, la especificación e implementación de entornos, el soporte a los procesos que determinan su funcionalidad y la definición de las opciones de interacción, supervisión y control por parte de los operadores, se realizan íntegramente mediante la formulación de modelos. Para dar soporte a esta capacidad, el modelo de referencia incluye un metamodelo que formaliza tales modelos.

Palabras clave: MDSE, meta-modelado, entorno de desarrollo.

1 Introducción

Se presenta *MDDE* (*Model-Driven Development Environment*), una concepción genérica para diseño de entornos de desarrollo de software basados en MDSE. Su objetivo es fomentar la aceptación, adopción y consolidación de MDSE entre los ingenieros software encargados del diseño e implementación de nuevos entornos de soporte a las metodologías que proponen, a los que añaden nuevos modelos de información, herramientas y/o procesos de desarrollo.

El principal componente de *MDDE* es un modelo de referencia (*MDDE framework*) que define las capacidades básicas, tanto funcionales como de interacción, que son comunes a cualquier entorno. Representa una abstracción que especifica sus elementos constituyentes, tanto conceptuales (caracterizados por el papel que juegan) como funcionales (caracterizados por los servicios que prestan y la interfaz que ofrecen), define la arquitectura con que se integran y los mecanismos de interacción entre ellos. Constituye un mapa conceptual independiente de cualquier implementación que proporciona al diseñador de sistemas software la información necesaria para trabajar con un entorno *MDDE* y al diseñador de entornos la vista arquitectural que dirige los elementos que

adfa, p. 1, 2011.

diseña. Aunque formulado de forma agnóstica respecto a plataforma, para su validación requiere ser implementado sobre una plataforma concreta. Así, dada una plataforma X, la implementación de *MDDE* sobre X se denomina *MDDE-X*, lo cual es ortogonal al dominio y/o metodología a que se pretenda dar soporte. Por ejemplo, podría hablarse de una implementación de *MDDE* sobre Eclipse y concebida para dar soporte al desarrollo de Sistemas de Tiempo Real Embebidos (STRE) basado en MAST [1], con lo cual se trataría del entorno *MDDE-Eclipse-MAST*.

El modelo de referencia puede considerarse desde tres puntos de vista: i) El *punto de vista estructural* contempla los elementos conceptuales que lo constituyen, las relaciones entre ellos y su formalización como metamodelo; ii) el *punto de vista funcional* lo aborda definiendo los elementos que constituyen el motor interno del entorno y las interfaces que utiliza; y iii) *el punto de vista de la implementación*. En esta comunicación se describen principalmente los aspectos estructurales y, como prueba de concepto, también se describe de forma simplificada el ejemplo de implementación *MDDE-MinimalMAST2*. Por motivos de espacio, la exposición del punto de vista funcional queda omitida, aunque en [2] puede encontrarse su estudio en profundidad, incluyendo la idoneidad de Eclipse como plataforma de soporte funcional.

El resto del artículo tiene la siguiente estructura. Tras esta sección introductoria, la sección 2 presenta una visión general de la concepción *MDDE*, mientras que las siguientes secciones abordan el modelo de referencia de *MDDE* desde el punto de vista estructural (sección 3) y de implementación (sección 4). La sección 5 analiza las posibilidades de *MDDE* para el diseñador de entornos. Finalmente la sección 6 aborda trabajos relacionados existentes en la bibliografía y la sección 7 esboza algunas conclusiones y líneas de trabajo futuras.

2 Visión general de *MDDE* y caracterización de los entornos

2.1 Objetivos

Para el diseño de entornos de desarrollo de software bajo *MDDE* se requiere definir las funcionalidades básicas que éstos han de proporcionar: Servir de repositorio (persistente o temporal) de los modelos que contienen la información (introducida o resultante) relativa a los procesos de desarrollo de software, dar soporte a las herramientas con las que se maneja tal información y proporcionar recursos de interacción para que el desarrollador pueda controlar la ejecución de dichos procesos de desarrollo. Asimismo, en la concepción *MDDE* se han considerado otros objetivos, como integrar de forma natural los múltiples aspectos y dominios que conciernen al diseño de un sistema software, proporcionar una estrategia estandarizada e independiente del dominio para acceso a la información y a los procesos, y facilitar el futuro mantenimiento, extensión y portabilidad de los entornos a través de la modularidad, estandarización y reutilización de sus elementos. Todo ello, bajo la consideración de que el entorno de desarrollo va a ser diseñado e implementado por ingenieros software expertos en muy diferentes dominios pero que habitualmente no son expertos en tecnologías MDSE.

2.2 Tipos de entornos *MDDE*

La concepción *MDDE* es única, pero el contenido nativo con que se distribuye un entorno *MDDE* determina su propósito así como su capacidad de evolución y enriquecimiento futuro. Se distingue entre *entorno MDDE especializado, extensible* y de *propósito general*. Cada entorno *especializado* tiene por objetivo dar soporte, a través del contenido nativo con que es distribuido, a uno o varios dominios en el desarrollo de software. Sin embargo, los de este tipo no están dotados de los recursos para incorporar nuevos metamodelos y herramientas, por lo que no puede ampliarse su funcionalidad. En cambio, los entornos *extensibles* se distribuyen con recursos nativos que permiten crear nuevos entornos especializados, con los elementos funcionales considerados apropiados para los usuarios finales. Por último, los entornos *de propósito general* se distribuyen dotados tanto de contenido nativo de soporte al desarrollo de software en un cierto conjunto de dominios como de los recursos necesarios para que expertos en nuevos dominios puedan crear nuevos metamodelos y herramientas de soporte a nuevos procesos de desarrollo, extendiendo así persistentemente la funcionalidad del entorno a los nuevos dominios.

2.3 Fundamento operacional

MDDE considera que el operador que usa un entorno para desarrollar proyectos software ejecuta, de forma supervisada, un cierto conjunto de *procesos*, cada uno de los cuales consiste a su vez en la ejecución secuencial o iterativa de operaciones más básicas denominadas *tareas*. De acuerdo al espíritu MDSE, los procesos se formulan como modelos que describen la secuencia de tareas constituyentes y especifican su naturaleza, siendo el entorno el encargado de interpretarlos y ejecutarlos, siempre bajo la supervisión del operador. A su vez, las tareas se formulan como modelos que describen su naturaleza, los modelos sobre los que operan, las transformaciones involucradas, la información ofrecida al operador para la toma de decisiones y las opciones de control con las que supervisa, valida y dirige su ejecución. Así, la especificación y diseño de un entorno va a consistir básicamente en la elaboración de modelos, y no en el desarrollo de su código de implementación. Para dar soporte a esta capacidad, el modelo de referencia incluye un metamodelo que formaliza tales modelos. Como prueba de concepto se ha diseñado e implementado una herramienta que los interpreta, y en base a ellos, genera automáticamente los recursos del entorno diseñado.

3 Modelo de referencia: elementos conceptuales y estructura

Desde un punto de vista estructural, el modelo de referencia de *MDDE* puede describirse en dos niveles. En primer lugar (subsección 3.1), descripción de sus elementos conceptuales, en base a sus atributos e interrelaciones, especificando para cada uno de ellos sus tipos derivados y su clasificación. En segundo lugar (subsección 3.2), la formulación del metamodelo que formaliza los aspectos estructurales del entorno conceptual y sirve de base para la descripción de cada entorno concreto.

3.1 Elementos conceptuales del *MDDE framework*

Modelos. La información manejada en un entorno *MDDE* está formulada como modelos conformes a metamodelos existentes en él. Según su función, se distingue entre modelos de: i) *dominio*, que contienen información sobre un aspecto de un sistema bajo desarrollo (SBD); ii) *herramienta*, que describen la operatividad, configuración o estatus de una herramienta en el entorno y iii) *interacción*, para describir la información que intercambian operador y entorno. Por último, también son modelos los *metamodelos* encargados de describir la información que contendrán los otros modelos.

Atendiendo al ciclo de vida en el entorno, se distingue entre modelos: i) *nativos*, incorporados al entorno durante su creación y por tanto distribuidos de inicio con él; ii) *de proyecto*, con información relevante para el proyecto en desarrollo y producidos bajo demanda del operador o por requerirlo la estrategia de desarrollo y iii) *temporales*, creados transitoriamente para manejar información interna dentro de un proceso del entorno. El conjunto de modelos nativos (dado su carácter, no modificables y no eliminables, aunque pueden ser supervisados o copiados por el operador), junto al resto de elementos nativos, constituye la base de la capacidad funcional y evolutiva de un entorno *MDDE*. En cuanto a los modelos de proyecto, pueden ser construidos manualmente o generados mediante la invocación de procesos en el entorno, y siempre son persistidos en su espacio de datos. El operador o los procesos pueden modificarlos o eliminarlos, pero si esto no ocurre, el entorno los salva persistentemente cuando se cierra el proyecto y los recupera cuando se abre de nuevo. Por último, si la funcionalidad del proceso que genera modelos temporales lo permite, pueden ser supervisados y modificados por el operador en aquellas fases de su ciclo de vida en que son accesibles para él, pero siempre son destruidos cuando finaliza el proceso.

Herramientas *MDDE* (*mTool*). Una *mTool* es un proceso u operación de alto nivel ofrecido por un entorno *MDDE* para algún objetivo de desarrollo o de gestión propia, que enriquece, supervisa o procesa la información contenida en él, y que es invocada explícitamente por el operador desde el propio entorno. Toda *mTool* se compone de una secuencia de actividades más elementales (*tareas*) definidas de forma individual para facilitar su reutilización en otras *mTools*.

Las *mTools* permiten al operador incorporar al entorno, de forma asistida, información relativa al SBD, introduciendo nuevos modelos que enriquecen la información ya existente y presentar la información del SBD disponible en el entorno de acuerdo con el punto de vista que incorpora la *mTool* utilizada, así como procesarla de acuerdo con estrategias de transformación, integración o análisis incorporadas por ella. Permiten además gestionar, también de forma asistida, los modelos del entorno, organizando, persistiendo o eliminando los elementos dentro de él o importar y exportar la información del SBD desde o hacia otros formatos diferentes a los del entorno.

Toda *mTool* se formula a través de un modelo, por lo que su incorporación a un entorno consiste en registrar dicho modelo. Algunas *mTools* son nativas, esto es, sus modelos se han registrado en la creación del entorno y permanecen inalteradas durante todo su ciclo de vida, proporcionándole una funcionalidad operativa básica. Por otro lado, el operador (en el rol de diseñador de entornos) puede definir nuevas *mTools* aportando los correspondientes modelos descriptivos o modificando los de otras ya

existentes. El modelo de una *mTool* describe su: i) funcionalidad, es decir, la secuencia de *tareas* de que se compone; ii) contexto de invocación, esto es, el elemento del entorno desde donde puede ser invocada, bien directamente (menú o botón) o bien por selección contextual; iii) opciones de configuración y iv) información de estado que ha de ir generando mientras se está ejecutando. Los parámetros de configuración están definidos como un submodelo que especifica su identificación, su tipo, valor que se les asigna (siempre tienen al menos valores por defecto) y si pueden ser modificados por el operador. Por su parte, el estado de ejecución se describe también como un submodelo cuyos valores son establecidos de acuerdo con los resultados que se obtienen en fase de ejecución, proporcionando información relevante respecto al éxito/fracaso de la ejecución de las *tareas* internas y respecto a los modelos que va generando la ejecución de la *mTool* en cuestión.

La ejecución de una *mTool* se realiza siempre bajo control y supervisión del operador. Tras la invocación, hay al menos tres puntos en los que se requiere su intervención: 1) Establecer y/o aceptar la configuración de lanzamiento; 2) lanzar la ejecución y 3) dar por concluida la ejecución (supone la eliminación de toda la información de estatus y modelos transitorios generados). Cuando éstos son los únicos puntos de intervención, se trata de ejecución en *modo continuo*, en contraposición a modo *paso a paso*, en que la ejecución requiere que el operador intervenga en la ejecución de cada *tarea* integrante. Como se verá posteriormente en esta misma subsección, para gestionar la ejecución de una *mTool*, tras ser invocada se despliega en el entorno un marco de interacción con los controles necesarios para ello y se puebla en base a su modelo descriptivo, mostrando la secuencia de *tareas* constituyentes.

Tareas *MDDE* (*mTask*). En el *MDDE framework*, el elemento operativo básico, representativo de alguna operación útil para el desarrollo de sistemas o para la gestión del propio entorno, se denomina *mTask*. Así, las *mTasks* son las actividades elementales que constituyen las *mTools* del entorno y se introducen con varios objetivos: i) Posibilitar la reutilización de operaciones compartidas por diferentes *mTools*; ii) ofrecer una interfaz de gestión homogénea que facilite la composición de actividades en los procesos y la interacción con los recursos del entorno y iii) constituir un adaptador para artefactos desarrollados con independencia de él (ver apartado siguiente).

Según su ciclo de ejecución, una *mTask* puede ser *atómica* o *interactiva*. Atómica significa que su ejecución se realiza únicamente en base a datos de configuración iniciales, por lo que se ejecuta sin requerir intervención del operador. En cambio, interactiva significa que sí requiere su intervención, tanto para gestionar información como para decidir las opciones de flujo de control. Por otro lado, según la forma en que las *mTasks* implementan su funcionalidad, pueden clasificarse como *nativas*, esto es, definidas en el entorno y que sólo requieren su invocación directa, o con funcionalidad definida mediante un modelo. Además, también se consideran *mTasks* de tipo adaptador de artefacto (ver apartado siguiente).

Artefactos *MDDE* (*mGadgets*). Son recursos software utilizados por las *mTools* pero que han sido desarrollados fuera del entorno, con independencia de su modelo de referencia. Normalmente, un *mGadget* proporciona una funcionalidad de procesamiento de modelos, presentación de información o de interacción con el operador y tanto su empleo por una *mTool* como su interacción con los recursos del entorno no se

realizan de manera directa, sino que se precisa de un adaptador que permita su invocación, configuración, manifestación de estatus e interacción con el operador.

Un *mGadget* puede ser interno (se ejecuta en el espacio de memoria del propio entorno en que se invoca) o externo (tiene que ejecutarse en un espacio de memoria distinto, bien en el mismo procesador que el entorno o en otro diferente – *mGadget* externo y remoto). La gestión de los de tipo externo y su interacción con el entorno se realizan a través de mecanismos de comunicación establecidos en el modelo de referencia de *MDDE* y proporcionados por la plataforma en que se ejecuta el entorno. Asimismo, para su invocación se requiere que en el espacio de memoria en que se ejecuta haya una aplicación *mGadgetLauncher* de lanzamiento de *mGadgets*.

La distinción internos / externos implica una distinción paralela entre las *mTasks* de tipo adaptador. Las de adaptación entre el entorno y uno interno proporcionan los modelos de entrada requeridos por el *mGadget* en el formato adecuado, almacenan en el entorno los modelos de resultados que éste genera y traducen los mensajes de control y estatus entre él y el entorno. Las de adaptación de uno externo requieren la intervención del servicio de comunicación de la plataforma así como la serialización de los modelos y mensajes intercambiados entre ésta y el *mGadget*.

Información no formalizada como modelos. El ámbito externo a un entorno *MDDE* es heterogéneo, por lo que ni se puede ni conviene mantener que la información ha de estar formulada como modelos conformes a metamodelos conocidos desde el entorno. Por ello, en *MDDE* se ha considerado dar soporte a la formulación de información mediante lenguajes textuales, posibilitando así su intercambio con otros entornos o sistemas que tienen su propio lenguaje específico. Con ello también se consigue un mecanismo sencillo para que colaboradores sin acceso al entorno aporten información o la reciban de él y además proporciona un medio perdurable si se prevé que su duración en el tiempo va a ser más prolongada que la supervivencia del propio entorno. Por ejemplo, puede contemplarse la importación o exportación de información textual de tipo XML formalizada a través de plantillas W3C-Schema o lenguajes específicos desarrollados mediante Xtext [3].

MDDE workbench. GUI de referencia que ha de implementar todo entorno *MDDE* para posibilitar la actuación del operador con la información contenida en el espacio de trabajo y con los procesos en el entorno. La **Fig. 1** lo esquematiza. Básicamente se compone de un área superior (secciones S1 y S2), una zona central (sección S3) y una región periférica (secciones S4, S5, S6 y S7). Cada una de estas últimas está formada por uno o más *marcos de interacción*, áreas gráficas con visores y controles complementarios que implementan un mecanismo de interacción con el operador. La *sección de gestión de elementos* (S4) presenta dos marcos de tipo explorador para organizar y acceder a la información (modelos, metamodelos, etc.) en el entorno. La *sección de información sobre elementos* (S5) presenta dos marcos para exponer información del elemento seleccionado en los marcos de S4. La *sección de información de la ejecución de herramientas* (S6) presenta dos marcos para exponer información propia de la ejecución de las *mTools*. Por último, la *sección de gestión de herramientas* (S7) proporciona al operador la capacidad de controlar la ejecución de una *mTool* previamente invocada, permitiéndole establecer su configuración, llevar a cabo de forma controlada su ejecución, supervisar el estatus de ejecución y finalizar o abortar la ejecución.

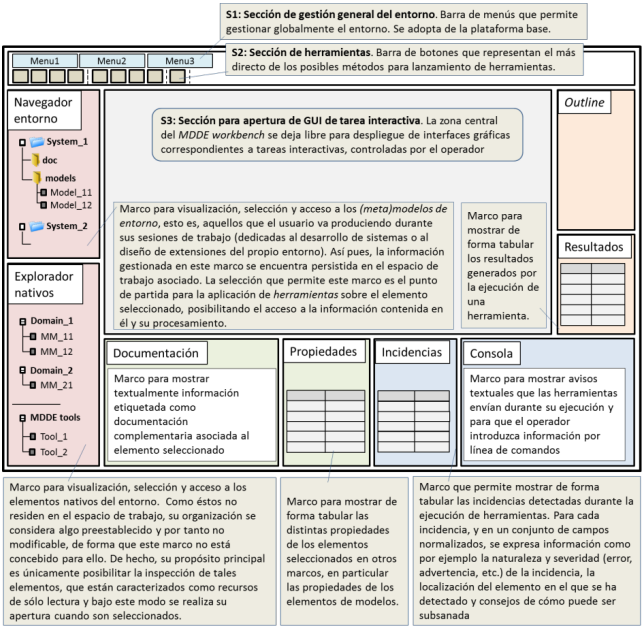


Fig. 1. Esquema del *MDDE workbench.*

La **Fig. 2** muestra en detalle el marco *Tool Outline*, cuya función es mostrar la constitución de una *mTool* (secuencia de *mTasks*) y controlar su ejecución.

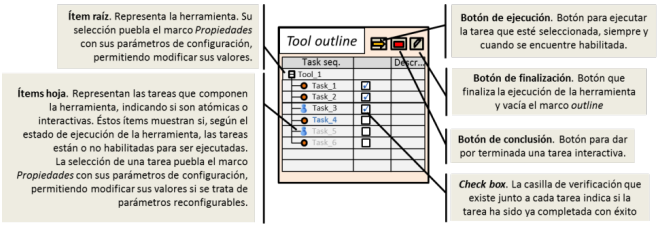


Fig. 2. Elementos de interacción del marco *Tool outline*

3.2 Metamodelado de la parte conceptual del *MDDE framework*

El metamodelo MDDE formaliza los aspectos estructurales del *MDDE framework*. Presenta un diseño que permite especificar un entorno *MDDE* mediante la formulación conjunta de varios modelos conformes a él, tal y como muestra la **Fig. 3**. Se trata de los modelos de formulación de las *mTools* en el entorno, que a su vez encapsulan el modelado de las *mTasks* constituyentes, los modelos de formulación de los *mTaskTypes* definidos en el entorno y los modelos del entorno, que representan propiamente un entorno pero su papel se reduce simplemente a ser un aglutinador de los modelos anteriores. Por *mTaskType* se entiende un ente parametrizado (configurable) cuya realización específica (asignación de valores a sus parámetros de configuración) da lugar a una *mTask* concreta. Así, las *mTasks* contenidas en los modelos de *mTools* son realizaciones de los *mTaskTypes* formulados mediante estos modelos.

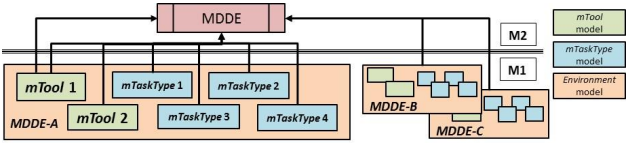


Fig. 3. Especificación de entornos *MDDE* mediante modelos

La **Fig. 4** muestra el núcleo del metamodelo *MDDE*. En función de todo lo expuesto anteriormente, la semántica de las clases mostradas es evidente. `TaskDescriptor` representa el concepto de *mTaskType*. `Task` representa el concepto de *mTask*, como realización concreta de una instancia *mTaskType* apuntada mediante la referencia `descriptor`. `Tool` representa el concepto de *mTool*. Su asociación `tasks` permite que una instancia suya albergue la secuencia de *mTasks* constituyentes, instancias de `Task`. Finalmente, `EnvironmentModel` representa el concepto de entorno. Una instancia suya referencia a través de las asociaciones `tools` y `taskDescriptors` a los conjuntos de *mTools* y *mTaskTypes* que forman la especificación de un entorno *MDDE*.

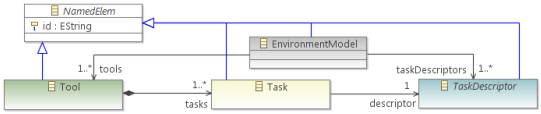


Fig. 4. Núcleo del metamodelo MDDE

Aunque pueda parecer que el metamodelo presenta una estructura convencional, con la clase `EnvironmentModel` como clase contenedor principal, sus asociaciones `tools` y `taskDescriptors` no exhiben carácter de composición. Esta clase simplemente da soporte al modelo aglutinador del resto de modelos. Así, `Tool` y `TaskDescriptor` también poseen rol de contenedor principal y cada modelo de *mTool* o de *mTaskType* cuenta respectivamente con una instancia suya en su raíz.

A continuación, la **Fig. 5** y la **Fig. 6** amplían la exposición del metamodelo MDDE, omitiendo aquí, por razones de espacio, la definición detallada de cada clase mostrada. La especificación completa del metamodelo, junto a su formulación Ecore, pueden encontrarse en http://www.istr.unican.es/members/cesarcuevas/phd/es/mdde.html.

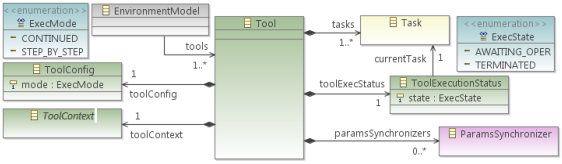


Fig. 5. Clase `Tool`

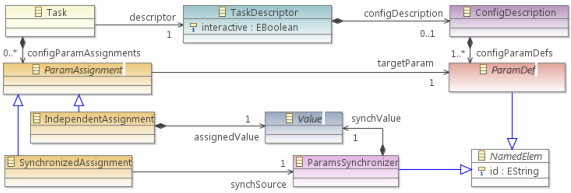


Fig. 6. Definición y asignación de parámetros

4 Ejemplo: Entorno *MDDE-MinimalMAST2*

Como ejemplo de entorno *MDDE* se propone *MDDE-MinimalMAST2*, concebido para dar soporte al análisis y diseño de STREs mediante la metodología MAST. Por motivos de espacio, no se presenta al completo. Su especificación completa junto a la formulación Ecore de los modelos mediante los que se representa así como su implementación sobre Eclipse (*MDDE-Eclipse-MinMAST2*) puede encontrarse en http://www.istr.unican.es/members/cesarcuevas/phd/es/mdde.html.

4.1 Visión general de MAST

MAST (*Modeling and Analysis Suite for Real-Time Applications*) es un entorno para diseño y análisis de STREs, desarrollado por el Grupo ISTR de la Universidad de Cantabria. Está básicamente constituido por una terna de modelos conceptuales (metamodelos MAST – actualmente completando su evolución a MAST-2 [4] –, MAST *Results* y MAST *Traces*) y un conjunto de herramientas que operan sobre modelos conformes.

4.2 Procesos del entorno *MDDE-MinimalMAST2*

El entorno ofrece tres procesos o *mTools*: i) Creación del modelo MAST-2 de un STRE; ii) análisis de planificabilidad y iii) simulación del comportamiento temporal. Dado un STRE modelado mediante MAST-2, la segunda permite aplicar diversos tipos de análisis de planificabilidad empleando las herramientas MAST-1.X, mientras que la tercera permite simular su evolución temporal empleando el simulador *JSimMAST2* [5], produciendo como salida un modelo MAST-2 *Results* y, opcionalmente, un modelo MAST-2 *Traces*. A continuación se expone en detalle esta tercera *mTool*.

La **Fig. 7** esquematiza su constitución en base a su secuencia de *mTasks*. Previamente a ejecutar la simulación propiamente dicha, el proceso contempla verificar la corrección del modelo MAST-2 de entrada (*mTask1*) y su compatibilidad con *JSimMAST2* (*mTask2*). La primera incluye la satisfacción de las restricciones de integridad del metamodelo MAST-2 y la segunda las específicas del simulador. A continuación, es necesario elegir un perfil para la ejecución de la simulación y en base a él transformar (*mTask3*) el modelo de entrada a uno de simulación (conforme al metamodelo SimMAST2). Llegado a este punto, el proceso consiste propiamente en ejecutar y controlar la simulación (*mTask4*), según el mismo perfil de simulación ya establecido.

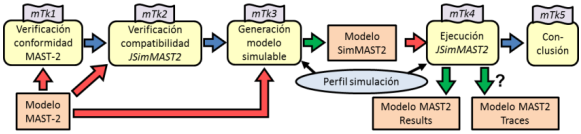


Fig. 7. *mTasks* constituyentes de la *mTool* de Simulación.

Como puede observarse, las *mTasks* expuestas se encuentran ligadas entre sí, pues el modelo de entrada a la *mTask1* lo es también a las *mTask2* y *mTask3*, el modelo de salida de la *mTask3* es modelo de entrada a la *mTask4* y el perfil de simulación establecido para la *mTask3* ha de coincidir con el establecido para la *mTask4*, pues según cuál vaya a ser éste, la generación del modelo simulable sigue unas directrices u otras.

4.3 Tipos de tarea

Para formalizar la constitución de las *mTools*, el entorno define cinco *mTaskTypes*: i) Verificación M2M de restricciones [6]; ii) Transformación MAST-2 → SimMAST2; iii) ejecución de simulación (lanzamiento del *JSimMAST2*); iv) ejecución de análisis de planificabilidad (lanzamiento de la herramienta externa MAST) y v) conclusión de proceso. A continuación se expone en detalle el primero.

M2M verification. Chequeo basado en M2M de si un modelo cumple un conjunto de restricciones especificadas sobre su metamodelo, ofreciendo como resultado un modelo de diagnóstico [6]. Las *mTasks* de este tipo son atómicas y están caracterizadas por poseer los siguientes parámetros de configuración (**Fig. 8**): localización del modelo a verificar, localización donde generar el modelo diagnóstico y el paquete de restricciones frente al que realizar la verificación.

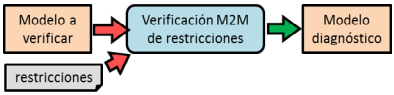


Fig. 8. *mTaskType* Verificación M2M de restricciones

Este *mTaskType* es un ejemplo paradigmático de reutilización, pues *mTasks* de este tipo pueden formar parte de multitud de *mTools* en las que antes de procesar modelos sea necesario realizar las verificaciones oportunas en cuanto a cumplimiento de restricciones. Es lo que sucede precisamente en las *mTools* de este entorno, pues mediante este *mTaskType* se da soporte, por ejemplo, a las *mTask1* y *mTask2* de la **Fig. 7**.

5 Herramientas y recursos para diseño de los entornos

MDDE define una variante del *workbench* orientada a los diseñadores de entornos específicos. Se llama *extension workbench* y su *layout* simplifica el de la versión original, prescindiendo de los marcos *Outline*, *Resultados* y *Consola*. Además, los contenidos de los marcos *Explorador de entorno* y *Navegador de elementos nativos* se ven modificados (**Fig. 9**), ya que ahora el ámbito de desarrollo es el propio dominio *MDDE* y al usuario le resultan de interés los *mTaskTypes* definidos en el entorno, de los cuales puede incluir realizaciones en las *mTools* que diseñe. Asimismo, también puede diseñar nuevos *mTaskTypes*. En el *Navegador de entorno* se ocultan cualesquiera elementos propios de sesiones de trabajo relativas al diseño de sistemas y en su lugar se muestra una sencilla estructura de contenedores en la que almacenar los modelos de *mTools* y *mTaskTypes* que el diseñador de entornos formule. En cuanto al *Explorador de elementos nativos*, se ocultan cualesquiera elementos propios del ámbito específico de desarrollo de sistemas y en su lugar se muestra el metamodelo MDDE y los *mTaskTypes* definidos en el entorno. Además, ahora el marco posee controles para añadir nuevas *mTools*, nuevos *mTaskTypes* o refrescar la visualización.

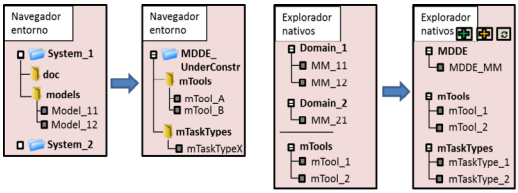


Fig. 9. Marcos *Explorador de entorno* y *Navegador de elementos nativos*

Creación de nuevas *mTools* y/o *mTaskTypes*. Se proporcionan los asistentes *NewTool* y *NewTaskType*, invocables respectivamente con los botones *Add mTool* y *Add mTaskType* del *Explorador de elementos nativos*. Su funcionamiento puede resumirse en que, mediante el cuadro de diálogo que emerge con la invocación, se introduce el

nombre de la *mTool* / *mTaskType* a desarrollar y tras ello se inicializa en memoria un modelo de *mTool* / *mTaskType* con la instancia `Tool` / `TaskDescriptor` contenedor principal adecuadamente inicializada. En el caso de *mTool*, la instancia `Tool` contiene dos instancias `Task`. La primera, *mTask_1*, no se encuentra asignada a ningún descriptor, mientras que la segunda, identificada *Termination*, es de tipo finalización de proceso. A continuación, el modelo generado se persiste en un fichero `*.tool.mdde` / `*.tasktype.mdde` con el nombre introducido por el usuario y ruta `workspace/MDDE_UnderConstr/tools | taskTypes`, donde `workspace` es la ruta del espacio de trabajo y la estructura de contenedores `MDDE_UnderConstr/tools | /taskTypes` se crea si no existe. Por último, se abre el modelo en el área de edición para que el operador complete su construcción.

Registro de las extensiones desarrolladas. Puesto que completar los modelos de *mTools* y *mTaskTypes* puede prolongarse un número indefinido de sesiones de trabajo, éstos permanecen persistidos en el espacio de trabajo, sin ser consideradas extensiones *MDDE* incorporadas de facto al entorno. Cuando se considera completada la formulación de un modelo, ha de procederse a su incorporación al entorno. Para ello, se definen las funcionalidades *Register mTool* y *Register mTaskType*, accesibles respectivamente mediante selección contextual de un modelo de *mTool* o *mTaskType* en el *Navegador de entorno*. El registro implica una fase preliminar de verificaciones (conformidad respecto al metamodelo MDDE y cumplimiento de restricciones) que, en caso de ser superada, conduce al registro propiamente dicho. Esto significa el traslado del modelo desde su ubicación provisional en el espacio de trabajo a la ubicación establecida para las extensiones *MDDE*, desapareciendo del *Navegador de entorno* y apareciendo en el *Explorador de elementos nativos*. A partir de entonces, en el caso de una *mTool*, ya se encuentra disponible para un diseñador de sistemas o, en el caso de un *mTaskType*, para su uso por el propio diseñador de entornos.

6 Trabajo relacionado

Un trabajo próximo al aquí expuesto es la metodología presentada en [7]. En ella se encuentran paralelismos con nuestro trabajo tanto en el objetivo final de facilitar el uso de MDSE como en la propia metodología en sí, basada en la ejecución semiautomática de flujos de actividades (*workflows*) [8]. Sin embargo, el trabajo se centra principalmente en los usuarios finales de entornos MDSE, tanto ingenieros de lenguajes como modeladores específicos de dominio, buscando agilizar sus actividades típicas. Nuestro trabajo, en cambio, está más orientado hacia la adopción de MDSE por parte de los ingenieros software a la hora de diseñar los entornos de desarrollo específicos de dominio que van a dar soporte a las metodologías propuestas por ellos, que típicamente no son expertos en tecnologías MDSE.

Los autores de [7] parten de una premisa básica: El éxito de MDSE depende tanto de que las herramientas aborden convenientemente las correspondientes tareas como de que permitan aumentar la productividad de los modeladores en sus actividades cotidianas. Sin embargo, la gran multitud de *frameworks* y herramientas de modelado exis-

tentes actualmente, aunque con funcionalidades basadas en fundamentos comunes, supone un importante hándicap para los usuarios. Éstos necesitan adaptarse a cada herramienta, pues cada una impone su propio proceso de desarrollo, el cual difiere ampliamente según la herramienta usada. Así, el objetivo planteado es incrementar la productividad de los modeladores en sus actividades habituales mediante la automatización de las tareas comunes realizadas con las herramientas MDSE actuales, desde operaciones simples (apertura, cierre o salvado de modelos) a tareas más complejas, como generación de artefactos para un DSL. También se considera la integración de tareas manuales. Para ello proponen una solución dirigida por modelos en la que se definen *workflows* reusables en forma de plantillas parametrizadas. Mediante la parametrización se consigue un mecanismo de reutilización para minimizar el número de *workflows* a ser creados, en base a que diversidad de tareas ocurren repetidamente en diferentes *workflows*. Puesto que la solución propuesta sigue el paradigma *model-driven*, la ejecución de *workflows* se modela completamente en forma de transformaciones de modelos, haciendo que sea reusable y portable en varias herramientas MDSE. Como parte de la metodología, se propone un DSL inspirado en los diagramas de actividad UML para el diseño de tales plantillas de *workflows*.

La metodología propuesta ha sido implementada en su propia herramienta AToMPM [9], aunque ha de poder ser implementada sobre diversidad de *frameworks* base, tanto soportando metamodelado a dos niveles como metamodelado profundo (*deep metamodelling*) [10, 11].

7 Conclusiones y trabajo futuro

El diseño de un entorno de desarrollo basado en MDSE no sólo requiere diseñar metamodelos que formalicen el soporte de la información y herramientas que lleven a cabo las transformaciones de ésta, sino también diseñar procesos que engloban conjuntos de modelos generados encadenando y/o iterando la aplicación de herramientas en el entorno bajo la supervisión del operador. Aunque concebir estos procesos sea responsabilidad del diseñador de entornos, que planea las estrategias con las que utilizar el entorno, su implementación en base a la infraestructura MDSE proporcionada por la plataforma de soporte al entorno queda, por su complejidad, fuera de su capacidad y conocimiento.

Para facilitar la tarea de este agente, se propone *MDDE*, una concepción genérica de entornos basados en MDSE que incluye la definición de un modelo de referencia para el diseño de entornos y de un conjunto de recursos de soporte que facilitan al experto diseñador de entornos su especificación e implementación. Siempre que éste acepte el modelo de referencia, los procesos se formulan como modelos que describen los modelos y herramientas participantes y las interacciones requeridas con el operador. Estos modelos descriptivos de procesos son interpretados por una herramienta ofrecida por el entorno, permitiendo su ejecución.

Por el momento, la validación de *MDDE* se encuentra en una etapa inicial, pues sólo se ha abordado una implementación, a modo de prueba de concepto, sobre la plataforma

Eclipse y orientada al ámbito de los STREs haciendo uso de las herramientas del entorno MAST. Es necesario seguir abordando otros ámbitos de desarrollo de sistemas software que permitan identificar puntos de extensión para *MDDE*, ampliándola en consecuencia y realizando las correspondientes implementaciones.

Agradecimientos. Este trabajo ha sido financiado parcialmente por el Gobierno de España con referencia TIN2014-56158-C4-2-P (M2C2).

Referencias bibliográficas

1. M. González Harbour, J. J. Gutiérrez García, J. L. Medina, J. C. Palencia, J. M. Drake, J. M. Rivas, P. López Martínez and C. Cuevas, "MAST: Bringing response-time analysis into real-time systems engineering," in *Workshop on Real-Time Systems: The Past, the Present, and the Future,*Anonymous York (UK): CreateSpace Independent Publishing Platform, 2013, pp. 42-59.
2. C. Cuevas, "Metaherramientas MDE para el diseño de entornos de desarrollo de sistemas distribuidos de tiempo real," 2016.
3. M. Eysholdt and H. Behrens, "Xtext: Implement your language faster than the quick and dirty way," in *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion,* 2010, pp. 307-309.
4. C. Cuevas, J. M. Drake, P. López Martínez, J. J. Gutiérrez García, M. González Harbour, J. L. Medina and J. C. Palencia, "MAST 2 Metamodel," 2012.
5. C. Cuevas, P. López Martínez and J. M. Drake, "JSimMAST: Simulador de sistemas de tiempo real diseñados con paradigmas avanzados," in Congreso Español de Informática (CEDI), III Simposio de Sistemas de Tiempo Real, Valencia (Spain), 2010, pp. 69-74.
6. C. Cuevas, P. López Martínez and J. M. Drake, "Model-driven approach for verifying conformity of models in the presence of constraints," in 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD), Rome (Italy), 2016, pp. 455-466.
7. M. A. Gamboa and E. Syriani, "Automating activities in MDE tools," in 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD), Rome (Italy), 2016, .
8. N. Russell, A. H. Ter Hofstede and N. Mulyar, "Workflow Controlflow Patterns: A revised view," 2006.
9. E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. Van Mierlo and H. Ergin, "AToMPM: A web-based modeling environment." in *Demos/Posters/StudentResearch@ MoDELS,* 2013, pp. 21-25.
10. J. De Lara and E. Guerra, "Deep meta-modelling with MetaDepth," in *Objects, Models, Components, Patterns*Anonymous Springer, 2010, pp. 1-20.
11. A. Rossini, J. de Lara, E. Guerra, A. Rutle and U. Wolter, "A Formalisation of Deep Metamodelling," *Formal Aspects of Computing,* vol. 26, pp. 1115-1152, 2014.

Desarrollo Eficiente de Lenguajes Específicos de Dominio para la Ejecución de Procesos de Minería de Datos

Alfonso de la Vega, Diego García-Saiz, Marta Zorrilla, y Pablo Sánchez

Dpto. Ingeniería Informática y Electrónica
Universidad de Cantabria, Santander (España)
{alfonso.delavega, diego.garcia,
marta.zorrilla, p.sanchez}@unican.es

Resumen Aunque las técnicas de minería de datos están consiguiendo cada día una mayor popularidad, su complejidad impide que sean aún utilizables por personas sin un sólido conocimiento en las mismas. Una posible solución, ya explorada por los autores de este artículo, es la construcción de Lenguajes Específicos de Dominio que proporcionen una serie de primitivas de alto nivel para la ejecución de procesos de minería de datos. Dichas primitivas sólo hacen referencia a terminología propia del dominio analizado, enmascarando detalles técnicos de bajo nivel. No obstante, la construcción de un lenguaje específico de dominio puede ser un proceso costoso. Este artículo muestra cómo reducir los tiempos de desarrollo de estos lenguajes de análisis mediante la reutilización de partes comunes de estos DSLs.

Palabras clave: Lenguajes Específicos de Dominio · Desarrollo de Software Dirigido por Modelos · Minería de Datos

1. Introducción

El uso de técnicas de análisis de datos está adquiriendo cada vez una mayor popularidad. Este incremento se debe en parte a la mayor presencia en nuestra vida de sistemas informáticos, los cuales almacenan datos acerca de nuestras actividades que, convenientemente analizados, pueden ayudar a mejorar ciertos procesos y negocios. Un ejemplo de ello es cómo la plataforma de vídeo bajo demanda *Netflix* utiliza los datos de actividad de sus usuarios para decidir cuáles deberían ser las siguientes series y películas a producir [15].

No obstante, el análisis de estos datos es una tarea especializada que no está al alcance de cualquier profesional, ya que requiere de un conocimiento sólido y detallado de una serie de técnicas y algoritmos especializados. Prueba de ello es la creciente demanda de perfiles profesionales en *Ciencia de Datos*, los cuales son difíciles de encontrar y suelen tener un alto coste asociado [13].

Para solventar este problema, nos planteamos la idea de construir lenguajes de consulta, mediante primitivas de alto nivel y terminología propia de un dominio concreto. Estos lenguajes permitirían a las personas responsables de un

proceso especificar tareas que hicieran uso de técnicas de minería de datos, con el objetivo de extraer información que permitiese mejorar estos procesos, y haciendo innecesario que los usuarios deban poseer conocimientos acerca de dichas técnicas de minería.

En un trabajo anterior [16] exploramos la posibilidad de crear estos Lenguajes Específicos de Dominio (en adelante DSLs, por sus siglas en inglés *Domain Specific Languages*). Concretamente, se desarrolló un DSL para el ámbito educacional, que permitía analizar el rendimiento de un curso o asignatura a partir de datos recopilados por la plataforma de *e-learning* utilizada durante el desarrollo de dicho curso.

Aunque en [16] se demostró que es factible, bajo ciertas condiciones, crear este tipo de DSLs, se constató que el coste asociado a su desarrollo desde cero puede ser elevado. No obstante, en dicho trabajo se apreció la posibilidad de reutilizar ciertos elementos para el desarrollo de DSLs similares, lo que ayudaría a reducir sus tiempos de desarrollo y, por consiguiente, su coste.

Siguiendo esta idea, este trabajo presenta una infraestructura para el desarrollo de DSLs de análisis de datos, que trata de explotar dichas posibilidades de reutilización. El objetivo final es crear un marco de trabajo que permita desarrollar estos DSLs de una forma más eficiente que partiendo desde cero cada vez.

Para comprobar la efectividad de nuestra propuesta, se utilizó dicha infraestructura para generar dos DSLs: (1) el que se había creado manualmente para el ámbito educacional [16]; y (2) otro para el análisis de datos médicos relacionados con la diabetes. Se constató que era posible la reutilización de partes significativas de estos DSLs y que dicha reutilización ayudaba a reducir los tiempos asociados a su desarrollo.

Tras esta introducción, este artículo se estructura como sigue: La sección 2 describe la motivación existente tras este trabajo, así como trabajos previos relacionados que han intentado acercar la minería de datos a usuarios no expertos. La sección 3 describe la infraestructura creada para facilitar el desarrollo de DSLs para análisis de datos. Por último, la sección 4 concluye este trabajo, resaltando las principales conclusiones obtenidas y comentando posibles trabajos futuros.

2. Motivación y Antecedentes

Esta sección describe la motivación tras este trabajo. Primero, se describe por qué estamos interesados en desarrollar DSLs para el análisis de datos y en segundo lugar por qué es conveniente crear una infraestructura que facilite el desarrollo de dichos DSLs.

2.1. Por qué DSLs para Minería de Datos

El trabajo que aquí presentamos se basa en la siguiente hipótesis: las técnicas actualmente disponibles para el análisis de datos, y en especial las relacionadas

con la minería de datos, solamente pueden ser utilizadas por personas con un sólido conocimiento en las mismas. Si se desea que profesionales que carezcan de dicho conocimiento especializado utilicen o se beneficien de estas técnicas, es necesario el desarrollo de sistemas que oculten su complejidad.

Para verificar que esta hipótesis es cierta, realizamos una revisión sistemática de la literatura siguiendo las directrices de Kitchenham [9] y Wohlin [7]. En dicha revisión se analizaron una serie de herramientas y trabajos cuyo objetivo era acercar las técnicas de minería de datos a los usuarios. Una descripción detallada de los aspectos formales de dicha revisión sistemática, por razones de espacio, queda fuera del ámbito de este artículo. Como resultado de la misma, se identificaron cuatro grupos principales de trabajos y herramientas, los cuales describimos a continuación.

Soluciones Ad-Hoc para Problemas Concretos Este grupo se compone de aplicaciones diseñadas para resolver un problema de análisis de datos específico dentro de un dominio muy concreto. Por ejemplo, Kamsu et al. [8] desarrollaron un sistema para extraer patrones dentro de datos provenientes de hemodiálisis. En estos casos, el problema es bien conocido desde el inicio y las técnicas de minería de datos se seleccionan y optimizan para ese problema concreto. A continuación, se construye una interfaz amigable que permita a los usuarios, expertos en el dominio del problema pero no en técnicas de análisis de datos, ejecutar dichas técnicas. Dependiendo del trabajo, el usuario puede en cierta medida refinar los resultados obtenidos o modificar la tarea de análisis de datos.

El principal inconveniente de estos trabajos es su coste asociado, ya que implican desarrollar una aplicación desde cero. Además, las aplicaciones desarrolladas no son fácilmente adaptables a otros problemas o dominios.

Herramientas de Minería Basadas en Workflows Este conjunto agrupa una serie de herramientas para la definición de procesos de minería de datos mediante una serie de bloques reutilizables. Estos bloques representan algoritmos y tareas típicas de los procesos de minería de datos y se pueden combinar de forma rápida y eficiente para definir procesos completos de minería de datos. *Rapidminer* [1] y *Weka* [6] son ejemplos de este tipo de aplicaciones. Estas herramientas ayudan a reducir los tiempos de desarrollo de los procesos de minería, pero no están orientadas a usuarios no expertos. Por ejemplo, cada bloque requiere la correcta configuración de una serie de parámetros internos, no estando esa tarea al alcance del usuario medio, por lo que se precisa de la contratación de personal experto.

Metodologías Orientadas a Usuarios No Expertos Esta categoría contiene una serie de metodologías que tratan de asegurar que las aplicaciones desarrolladas puedan ser utilizadas por usuarios sin conocimientos de minería de datos.

Un ejemplo de estas metodologías es el trabajo de Zorrilla y García [17] donde se propone, en primer lugar, construir servicios que encapsulen procesos

de minería de datos para un dominio concreto. A diferencia de los bloques de la categoría anterior, se busca que estos servicios sean autoconfigurables [3]. A continuación, se desarrollarían una serie de interfaces web que permitirían a los usuarios no expertos seleccionar los conjuntos de datos a analizar, invocar dichos servicios y visualizar los resultados.

Estas metodologías siguen precisando de la contratación de expertos que las apliquen, aunque en muchos casos se intenta favorecer la reutilización de elementos. Además, en ciertas ocasiones, podría ser difícil adaptar las aplicaciones desarrolladas a nuevas situaciones. Por ejemplo, una aplicación para analizar datos de estudiantes podría no ser utilizable para analizar datos relativos a otras entidades del dominio, tales como exámenes o actividades en el aula.

Herramientas Genéricas Orientadas a Usuarios No Expertos Este grupo aglutina herramientas que proporcionan al usuario no experto comandos o primitivas de alto nivel que le permiten ejecutar ciertas tareas de minería de datos sin necesidad de poseer sólidos conocimientos sobre las mismas. En este grupo destacaríamos *Oracle Predictive Analytics* [4]. Esta herramienta proporciona una serie de procedimientos al estilo SQL que permiten extraer cierta información de los datos contenidos en una tabla de una base de datos (alojada en Oracle). Por ejemplo, el usuario puede utilizar el comando `EXPLAIN` para intentar averiguar qué valores de los atributos de la tabla producen un cierto valor en una columna determinada. De este modo, usando este comando se podría averiguar por qué ciertos estudiantes tienen la columna `Calificación` con el valor `Suspenso`.

Una ventaja inicial es que no es necesaria la contratación de expertos para ejecutar estos comandos (aunque podrían requerirse otras cuestiones, como familiaridad con Oracle, por ejemplo). La gran ventaja de esta técnica es que estos comandos no precisan ser modificados cuando se utilizan para nuevos datos. Por ejemplo, el comando `EXPLAIN` es el mismo con independencia de la tabla sobre la que se aplique. Por tanto, estos comandos sí podrían aplicarse a datos de estudiantes, exámenes o actividades en aula, siempre que estuviesen disponibles en sus correspondientes tablas. Sin embargo, esta genericidad, tiene una penalización sobre la precisión de los resultados, ya que las técnicas subyacentes utilizadas no están convenientemente optimizadas para cada dominio.

Tras analizar el estado del arte, decidimos explorar la siguiente idea: En un primer lugar, encapsularíamos diversos procesos de minería de datos en una serie de componentes reutilizables. El objetivo inicial era que estos componentes fuesen reutilizables para problemas y datos procedentes de diversos dominios, al estilo de *Oracle Predictive Analytics*. No obstante, estos componentes deberían poderse optimizar para un problema concreto si así se desease, tal como en Zorrilla y García [17]. En segundo lugar, para que estos componentes fuesen utilizables por usuarios no expertos, desarrollaríamos DSLs, al estilo de *Oracle Predictive Analytics*, pero adaptados a las necesidades de cada usuario. Estos DSLs tendrían que permitir plantear un problema de minería de datos utilizando primitivas de alto nivel y una jerga cercana al dominio del usuario. Se busca

```
00 show_profile of Students;
01 show_profile of Students with courseOutcome=fail;
02 find_reasons_for courseOutcome=fail of Students;
```

Figura 1. Ejemplo de consultas para el ámbito educacional.

además que la forma de plantear dichos problemas sea lo más flexible posible. Las sentencias escritas utilizando el DSL se transformarían en código de bajo nivel encargado de invocar los componentes reutilizables para dar respuesta al problema planteado por el usuario.

Esta idea fue inicialmente explorada mediante la creación de un DSL, más un conjunto de procesos de minería de datos, para el ámbito educacional. La siguiente sección describe dicho trabajo.

2.2. Un DSL para el Análisis de Datos Educacionales

Para explorar nuestra idea inicial, desarrollamos un DSL para el análisis de datos provenientes del dominio educacional [16]. El objetivo de este DSL era que los profesores pudiesen explotar los datos que plataformas de aprendizaje como Moodle [11] recopilan acerca del desarrollo de sus asignaturas, con el objetivo de extraer de dichos datos información que pudiese ayudar a mejorar el rendimiento las asignaturas.

Concretamente, tratamos de resolver dos tareas de análisis. La primera de ellas consistía en identificar los diferentes tipos de perfiles que existían dentro de un conjunto de datos. Este análisis permitiría, por ejemplo, dividir a los estudiantes de un curso en grupos con características comunes.

La segunda de ellas trataba de encontrar las causas por las cuales se producía cierto fenómeno. Por ejemplo, por qué los estudiantes no superaban el curso, por qué lo abandonaban o por qué dejaban de entregar una determinada tarea, entre otras cuestiones.

Para ambas tareas, creamos código Java que utilizaba una serie de algoritmos de minería de datos proporcionados por la plataforma *Weka* [6]. Este código se parametrizó para que pudiese ser generado mediante plantillas.

A continuación, se desarrolló un DSL para invocar dichos procesos reutilizables. La Figura 1 muestra ejemplos de consultas escritas en dicho DSL[1].

La primera consulta (Figura 1, Línea 00) permite al profesor dividir el conjunto de estudiantes de su curso en grupos que compartan características similares. Para ello aplica la primitiva `show_profile` a la entidad `Students`.

La segunda consulta (Figura 1, Línea 01) sería similar a la primera, pero restringe los datos de entrada, aplicando un filtro, a aquellos estudiantes que no hayan superado la asignatura. La idea en este caso es caracterizar grupos de estudiantes que no superan la asignatura.

[1] El DSL se presentó inicialmente en un foro de ámbito internacional, y es por ello por lo que sus términos están en inglés.

La tercera consulta (Figura 1, Línea 02) trata de encontrar las causas por las que los estudiantes no superan el curso. Para ello hace uso de la primitiva `find_reasons_for`, la cual requiere que se le proporcione un fenómeno a explicar (`courseOutcome=fail`) y una entidad (`Students`) a analizar.

Este DSL se implementó definiendo su correspondiente metamodelo en Ecore y su sintaxis textual con ayuda de *Xtext* [5]. La semántica quedaba definida mediante la generación, a partir de las consultas, de código Java que invocaba los correspondientes algoritmos de la plataforma Weka. La generación de código se implementó utilizando las herramientas facilitadas por la suite de ingeniería de modelos *Epsilon* [12].

Como puede apreciarse, el DSL sólo contiene una serie de primitivas de alto nivel (`find_reasons_for`, `show_profile`) y referencias a entidades procedentes del dominio del usuario (`Student`) o a atributos de dichas entidades (`courseOutcome`).

A partir del nombre de una entidad, el lenguaje debería ser capaz de recuperar el conjunto de datos a analizar. Por tanto, debe existir una correspondencia entre estos nombres y los conjuntos de datos disponibles para su análisis. En nuestro caso, dicho nombre debe hacer referencia al nombre de un fichero de datos en formato ARFF, que es el formato que utiliza la plataforma Weka [6]. Cómo se obtiene la información a procesar y se almacena en dichos ficheros ARFF queda fuera del ámbito de este artículo.

Para fomentar su usabilidad, el DSL creado trata de minimizar los errores que pudiese cometer el usuario a la hora de escribir una consulta. Ello requiere: (1) que se eviten tanto como sea posible elementos libres; y (2) que se ofrezcan tantas funciones de autocompletado o asistencia como sea necesario. Por ejemplo, tal como se explicó anteriormente, como nombre de una entidad no debería ser posible escribir cualquier cadena, ya que este nombre debe corresponderse con el de un fichero de datos disponible. Por tanto, el DSL debería: (1) verificar esta restricción antes de compilar la consulta; y (2) ofrecer el conjunto de opciones disponibles al usuario durante la especificación de una consulta.

2.3. Problemas a Resolver

Tras la realización de dicho trabajo se identificaron dos problemas a resolver: (1) resultaba difícil en determinados contextos crear procesos de minería de datos que fuesen realmente reutilizables para diferentes dominios; y (2) el desarrollo de un DSL como éste desde cero puede convertirse en un proceso costoso que requeriría la contratación de expertos en DSLs. En este caso, estaríamos simplemente reemplazando el problema de tener que contratar expertos en minería de datos por el problema de tener que contratar expertos en el desarrollo de DSLs.

Este trabajo se centra en la resolución del segundo de estos problemas. El primer problema es un problema recurrente en minería de datos, que obliga a que, en muchas ocasiones, si se quieren obtener resultados precisos, los procesos tengan que ser diseñados ad-hoc para resolver un problema concreto y bien determinado. Dicho problema queda fuera del ámbito de este artículo, y será objeto de estudio en trabajos futuros. Por tanto, en adelante, asumiremos como

hipótesis que existen procesos de minería de datos reutilizables para diferentes dominios.

Con relación al segundo problema, se advirtió que ciertas partes de este DSL podían ser fácilmente reutilizables para el desarrollo de DSLs para nuevos dominios. Por ejemplo, la sintaxis textual concreta sería prácticamente la misma, ya que de dominio a dominio simplemente variarían los nombres que pueden ser utilizados como entidades del dominio. Igualmente, gran parte del proceso de generación de código debería ser reutilizable.

La solución presentada en este trabajo trata de explotar esta idea para crear un ecosistema de generación de estos DSLs para el análisis de datos. Para ello se ha rediseñado el DSL educacional, creando una infraestructura que permita un desarrollo más eficiente de DSLs para otros dominios. Dicha infraestructura se ha utilizado para el desarrollo de un DSL para el análisis de datos relacionados con pruebas de diabetes. Dicho dominio se describe a continuación.

2.4. Ejemplo: Análisis de Factores Relacionados con la Diabetes

Para el desarrollo del nuevo DSL se ha utilizado un conjunto de datos procedentes de pruebas realizadas para comprobar la existencia de diabetes en pacientes [14]. Por cada paciente, se recogen una serie de datos así como el resultado final de la prueba (positivo o negativo). Estos datos se pueden analizar para intentar averiguar si existen factores que puedan ser considerados como causas de una diabetes. Es decir, querríamos poder escribir una consulta en nuestro DSL como `find_reasons_for test_result = positive of Diabetes_Results`.

De igual forma, podría ser interesante plantear otras consultas, como averiguar si existen diferentes grupos o perfiles de pacientes con diabetes.

La siguiente sección describe cómo se ha creado la infraestructura anteriormente mencionada y cómo se ha utilizado para construir el DSL para el análisis de los datos de diabetes.

3. Una Infraestructura para el Desarrollo de DSLs para Minería de Datos

Para desarrollar una infraestructura que nos permitiese crear de manera eficiente un ecosistema de DSLs, en primer lugar precisábamos identificar qué elementos de dichos DSLs podían ser reutilizables.

Este proceso lo realizamos en dos fases: primero analizamos los componentes relacionados con la gramática y el editor del DSL; y luego aquellos relacionados con la generación del código encargado de procesar las consultas. Estos procesos se describen a continuación.

3.1. Desarrollo del Editor para el DSL

La gramática de nuestra familia de DSLs posee una componente claramente común. Esta componente abarca el conjunto de comandos o primitivas, tales

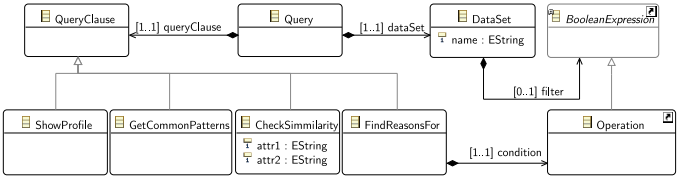


Figura 2. Sintaxis abstracta genérica.

como `find_reasons_for`, que el usuario puede utilizar para invocar tareas de análisis de datos. Por otro lado, existe una parte variable, que es la correspondiente a las referencias a entidades del dominio del usuario.

Estas referencias estaban incluidas en la gramática por dos razones: (1) evitar que el usuario pueda especificar consultas que se refieran a entidades inexistentes para las cuales no tenemos datos; (2) poder utilizar una serie de funciones de autocompletado que facilitasen al usuario la especificación de las consultas.

Teniendo estos factores en cuenta, la estrategia que adoptamos fue la siguiente: En primer lugar, se rediseñó la gramática para que fuese genérica para cualquier dominio. Con este fin, se liberó a la gramática de la responsabilidad de asegurar que las referencias a las entidades del dominio fuesen correctas. Esta responsabilidad se delegó en un *validador* externo, que se encargaría de asegurar que se satisfacen las restricciones relativas a las entidades del dominio. Para ello, el validador utilizaría como entrada un modelo externo a la gramática que especificase qué entidades y atributos están disponibles dentro de cada dominio. Las funciones de asistencia y autocompletado también harían uso de este modelo de entidades. De esta forma, para adaptar el DSL a un nuevo dominio, sólo sería necesario proporcionar un nuevo modelo de entidades y regenerar el editor.

La Figura 2 muestra parte de la sintaxis abstracta genérica creada para nuestra familia de DSLs. En ella se aprecia que cada consulta (`Query`) está definida por una primitiva (`QueryClause`) que determina la tarea de minería de datos a realizar. Esta consulta actúa sobre un determinado conjunto de datos (`Dataset`) el cual, de acuerdo con esta gramática, puede quedar definido por una cadena de caracteres arbitraria. Esto permite que la gramática sea utilizable para cualquier dominio. No obstante, utilizando sólo esta gramática, se podrían especificar consultas que no tuviesen ninguna relación con los conjuntos de datos disponibles.

Este problema es evitado con la ayuda del validador externo introducido que lleva a cabo estas comprobaciones. Para realizar esta comprobaciones, el validador utiliza un modelo de entidades, que debe ser conforme a un metamodelo de entidades. Dicho metamodelo se muestra en la Figura 3.

De acuerdo a este metamodelo, cada entidad tiene un nombre y un conjunto de atributos. Estos atributos pueden ser de diferentes tipos. Por ejemplo, pueden ser numéricos (`NumericalAttribute`) o enumerados (`NominalAttribute`). Los

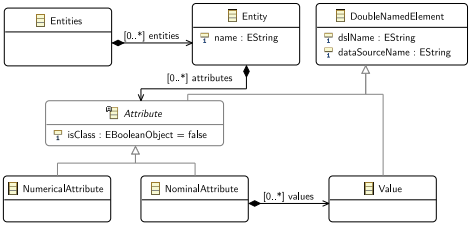


Figura 3. Metamodelo para la definición de información de las entidades analizadas.

```
01  @Check
02  def checkDataSetName(DataSet dataset) {
03    if (!entitiesProvider.entities.exists[
04          entity | entity.name.equals(dataset.name)
05        ]) {
06      error("Dataset with name '" + dataset.name + "' does not exist",
07          EdmdslPackage::eINSTANCE.dataSet_Name)
08    }
09  }
```

Figura 4. Función para validar el nombre de un `Dataset`.

atributos enumerados poseen asociado el conjunto finito de valores que pueden adoptar.

Además, cada atributo puede ser considerado como de clase (`isClass = true`) si sirve para particionar un conjunto de datos en clases relevantes para algún propósito. Esta distinción es relevante ya que en ciertas partes de nuestro DSL sólo se pueden utilizar atributos considerados de clase. Por ejemplo, puede interesar preguntar las causas por las que ciertos pacientes tienen diabetes (`find_reasons_for test_result=positive of Diabetes_Result`), pero no tendría mucho sentido preguntar las causas por las que un paciente reside en Londres (`find_reasons_for live_in=LONDON of Diabetes_Result`).

El validador externo se desarrolló utilizando las facilidades proporcionadas por Xtext. Para ello, especificamos una serie de funciones auxiliares en Xtend. A modo de ejemplo, la Figura 4 muestra la función que comprueba que el nombre de un `Dataset` introducido se corresponde con el nombre de una entidad de nuestro dominio. Como se puede observar, la función accede al modelo de entidades a través del objeto `entitiesProvider` (Figura 4, Línea 03) y comprueba si existe al menos una entidad cuyo nombre sea igual al del dataset.

De igual forma, para proporcionar funciones de asistencia y autocompletado, definimos otra serie de funciones auxiliares en Xtend. A modo de ejemplo, la Figura 5 muestra la función que carga los nombres de las entidades en el correspondiente menú contextual. Gracias a esta función, cada vez que el usuario

```
01  override public void completeDataSet_Name(EObject model,
02      Assignment assignment, ContentAssistContext context,
03      ICompletionProposalAcceptor acceptor) {

04    for (entity : entitiesProvider.entities) {
05      acceptor.accept(createCompletionProposal(entity.name, context))
06    }
07  }
```

Figura 5. Función que muestra los nombres disponibles para especificar un `Dataset`.

necesite escribir el nombre de un `Dataset`, aparecerá un listado con el conjunto de entidades disponibles.

Merece la pena destacar que estas funciones auxiliares de validación y asistencia al usuario serían reutilizables para diferentes dominios. Es decir, cada vez que queramos generar un DSL para un nuevo dominio, sólo necesitaríamos proporcionar el correspondiente modelo de entidades. El resto de elementos requeridos para el desarrollo del editor del DSL (sintaxis abstracta, sintaxis concreta, funciones auxiliares de validación, funciones auxiliares de asistencia y código de infraestructura) permanecerían sin modificar. Esto permite reducir los tiempos de desarrollo asociados al desarrollo de DSLs para nuevos dominios.

3.2. Desarrollo de los Generadores de Código

Tal como se comentó en la Sección 2.2, para ejecutar las consultas, éstas se transforman en código Java que invoca una serie de algoritmos de minería de datos proporcionados por la plataforma *Weka*. El proceso de transformación es completamente invisible para el usuario final, abstrayéndose por tanto de los detalles de bajo nivel. En este proceso, si se desea obtener mejores resultados o dependiendo las características de cada dominio, podría ser necesario modificar los algoritmos de minería de datos utilizados.

Por ejemplo, en base al dominio, para ejecutar la primitiva *show_profile* podría ser más adecuado utilizar como técnica de *clustering* el algoritmo *K-means*, mientras que en otros dominios el algoritmo *EM* (*Expectation Maximization*) podría generar mejores resultados.

Por otra parte, y aunque de momento no ha sido necesario, podría variar la plataforma de destino que proporciona las implementaciones de los algoritmos de minería de datos. Por ejemplo, podríamos optar por utilizar librerías presentes en *R* [2] en lugar de Weka.

Para ayudar a soportar esta variabilidad, introdujimos un metamodelo intermedio en el proceso de transformación para representar procesos de minería de datos de manera abstracta e independiente de cualquier plataforma. La parte izquierda de la Figura 6 muestra un fragmento de este metamodelo. Cada procedimiento se corresponde con un algoritmo de minería de datos que podría ser ejecutado. Estos algoritmos podrían requerir la configuración de parámetros adicionales. Por ejemplo, para llevar a cabo una tarea de clasificación, es necesario saber con respecto a qué atributo de un conjunto de datos deseamos realizarla

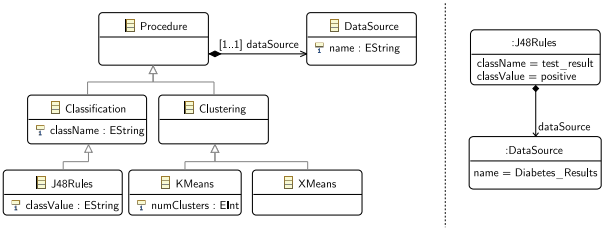


Figura 6. Izquierda: Metamodelo de un proceso abstracto de minería de datos; derecha: modelo del proceso de minería resultante de la transformación M2M.

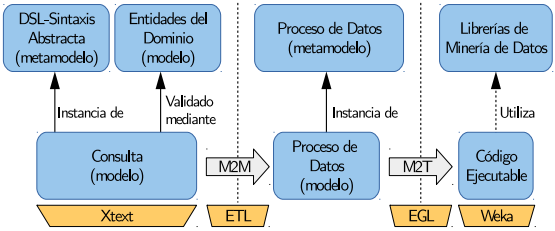


Figura 7. Transformaciones del proceso de ejecución de una consulta del DSL, indicándose en la parte inferior las herramientas involucradas en cada una de las etapas.

(atributo `className`). En el caso de querer utilizar el proceso *J48Rules*, que permite obtener las reglas que tratan de explicar por qué el atributo seleccionado toma un valor determinado, deberemos especificar además dicho valor de ese atributo (`classValue`).

Usando este metamodelo, una consulta de nuestro DSL se transforma, en primer lugar, en un modelo independiente de la plataforma que especifica qué algoritmos de minería de datos deben ser ejecutados para dar respuesta a esa consulta, tal como muestra la Figura 7. En segundo lugar, se realiza la generación de código para una plataforma concreta a partir de este modelo intermedio. La primera transformación modelo a modelo se realiza con el lenguaje *ETL (Epsilon Transformation Language)*, mientras que para la generación de código se emplea el lenguaje basado en plantillas *EGL (Epsilon Generation Language)*. Este proceso de transformación se ilustra a continuación para el ejemplo de los datos de diabetes.

Continuando con nuestro ejemplo anterior, la consulta `find_reasons_for test_result=positive of Diabetes_Result`, ya escrita en el DSL, se transformaría en el modelo mostrado en la parte derecha de la Figura 6. Este modelo indica que, para encontrar las razones por las cuales ciertos pacientes dan positivo en diabetes, se computará un árbol de decisión utilizando el algoritmo J48 [10]. Se

```
   // data obtention
01 DataSource source = new DataSource("data/diabetesResults.arff");
02 Instances ins = source.getDataSet();
03 ins.setClass(ins.attribute("test_result"));
   // rules generation
04 J48 j48 = new J48();
05 j48.buildClassifier(ins);
06 RuleSet ruleSet = j48.toRules();
07 ruleSet = ruleSet.filterByConsequent("positive");
   // results visualization
08 RulesVisualizer.show(ruleSet);
```

Figura 8. Código resultante de la transformación M2T sobre el modelo de minería.

```
IF (body_mass_index > 29.9 AND
    plasma_glucose_concentration > 157)
THEN result = tested_positive

Support: 11.979%, Confidence: 86.957%
```

Figura 9. Ejemplo del tipo de reglas obtenidas mediante la ejecución de la consulta.

analizarán los datos de los pacientes contenidos en el dataset `Diabetes_Result` utilizando como clase el atributo `test_result` para la construcción del árbol de decisión. A continuación, el algoritmo extraerá aquellas ramas que sirvan para explicar por qué ciertos pacientes padecen diabetes, es decir, aquellas cuyo valor predicho sea `positive`.

Una vez generado el modelo intermedio, (Figura 6, derecha), éste se usa como entrada para las plantillas de generación de código. En nuestro caso, las plantillas convertirían el modelo en código Java. La Figura 8 muestra parte del código generado. En primer lugar, se obtienen los datos mediante la carga del dataset especificado en un objeto de la clase *Instances* (Figura 8, Líneas 01-02). Esta clase Instances es una clase proporcionada por la plataforma Weka. Posteriormente, se ejecuta el algoritmo *J48*, tomando *test_result* como atributo de clase (Figura 8, Líneas 03-05). Dado que sólo nos interesan las reglas que retornen *positive* como resultado, aplicamos un filtro para quedarnos con aquellas reglas cuyo consecuente sea igual a *positive* (Figura 8, Líneas 06-07). Finalmente, se muestra las reglas obtenidas por pantalla (Figura 8, Línea 08).

La Figura 9 muestra, a modo de ejemplo, una de las reglas que el sistema mostraría como resultado al usuario que ha realizado la consulta para un dataset concreto. Esta regla determina que una causa de la diabetes puede ser poseer un índice de masa corporal superior a 29.9 y una concentración de glucosa en plasma mayor de 157. Además, para especificar el grado de certeza que podemos atribuir en dicha regla, se indica que este patrón aparecía en un 12 % de los casos y que, dentro de esos casos, prácticamente un 87 % padecía diabetes. Es decir, el patrón se da en un sector reducido de la población analizada, pero cuando aparece podría ser causa de una posible diabetes.

El uso del metamodelo intermedio proporciona las siguientes ventajas:

1. La introducción de cambios en el DSL afectaría solamente al proceso de transformación entre modelos, pero no a las plantillas de generación de código, siempre y cuando el cambio en el DSL no implicase la incorporación de nuevos elementos en el metamodelo de procesos de minería. Esto nos permite realizar pequeños cambios en la sintaxis del DSL sin tener que modificar las plantillas de generación de código.
2. Para cambiar el algoritmo de minería utilizado para computar una primitiva del DSL, sólo es necesario cambiar las transformaciones modelo a modelo ejecutadas. No sería necesario modificar las plantillas de generación de código, siempre que el nuevo algoritmo utilizado ya exista en el metamodelo de procesos de minería.
3. Si deseásemos cambiar la plataforma destino utilizada para ejecutar los algoritmos de minería de datos, sólo sería necesario modificar las plantillas de generación de código, permaneciendo las transformaciones modelo a modelo inalteradas.
4. El proceso de traducción de una consulta en un algoritmo de minería de datos se simplifica, ya que las transformaciones modelo a modelo están libres de la complejidad accidental introducida por las plataformas destino.

4. Sumario y Trabajos Futuros

La democratización de la minería de datos requiere del desarrollo de sistemas que permitan a usuarios no expertos beneficiarse de estas técnicas. Para alcanzar dicho objetivo, propusimos en un trabajo previo [16] la utilización de DSLs para minería de datos. Aunque los resultados iniciales eran prometedores, existían varios problemas a resolver. Uno de estos problemas era el coste asociado al desarrollo de un DSL.

Para aliviar dicho problema, en este trabajo se ha presentado una infraestructura de asistencia al desarrollo de estos DSLs. Esta infraestructura permite que se puedan reutilizar partes importantes de otros lenguajes de análisis de datos para la construcción de un DSL en un nuevo dominio. De igual forma, la infraestructura trata de minimizar los impactos producidos a la hora de introducir ciertos cambios en los DSLs. Esto permite reducir los tiempos de desarrollo de nuevos DSLs, reduciendo a la vez su coste.

Como trabajos futuros, se seguirá explorando la viabilidad de esta idea mediante la incorporación de nuevos dominios y/o problemas de análisis. Además, se estudiará la posibilidad de generar ciertos elementos de esta infraestructura. Por ejemplo, el modelo de entidades debería poderse generar sin demasiada dificultad desde los conjuntos de datos disponibles.

Agradecimientos. Este trabajo ha sido parcialmente financiado por el Gobierno de Cantabria (España) mediante el Programa de Personal Investigador en Formación Predoctoral de la Universidad de Cantabria y por el Gobierno de España en el proyecto TIN2014-56158-C4-2-P(M2C2).

Referencias

1. Rapidminer. https://rapidminer.com/
2. The R Project for Statistical Computing. https://www.r-project.org/
3. Balcázar, J.L.: Parameter-free Association Rule Mining with Yacaree. In: Actes Extraction et Gestion des Connaissances (EGC). pp. 251–254. Brest (France) (January 2011)
4. Campos, M., Stengard, P., Milenova, B.: Data-Centric Automated Data Mining. In: Fourth International Conference on Machine Learning and Applications (ICMLA'05). vol. 2005, pp. 97–104 (2005)
5. Eysholdt, M., Behrens, H.: Xtext: Implement Your Language Faster than the Quick and Dirty Way. In: Companion to the 25th Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications (SPLASH/OOPSLA). pp. 307–309. Reno/Tahoe (Nevada, USA) (October 2010)
6. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explorations Newsletter 11(1), 10–18 (June 2009)
7. Jalali, S., Wohlin, C.: Systematic literature studies: database searches vs. backward snowballing. In: Proceedings of the 2012 6th ACM_IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). pp. 29–38 (2012)
8. Kamsu-Foguem, B., Tchuenté-Foguem, G., Allart, L., Zennir, Y., Vilhelm, C., Mehdaoui, H., Zitouni, D., Hubert, H., Lemdani, M., Ravaux, P.: User-centered visual analysis using a hybrid reasoning architecture for intensive care units. Decision Support Systems 54(1), 496–509 (2012)
9. Kitchenham, B., Charters, S.: Guidelines for performing Systematic Literature Reviews in Software Engineering. Tech. Rep. EBSE 2007-001, Keele University and Durham University Joint Report (2007)
10. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)
11. Rice, W.: Moodle E-Learning Course Development. Packt Publishing (2006)
12. Rose, L.M., Paige, R.F., Kolovos, D.S., Polack, F.A.C.: Model Driven Architecture – Foundations and Applications: 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings, chap. The Epsilon Generation Language, pp. 1–16. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
13. Schrage, M.: Stop Searching for That Elusive Data Scientist. Harvard Business Review, https://hbr.org/2014/09/stop-searching-for-that-elusive-data-scientist/
14. Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., Johannes, R.S.: Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus. Proceedings of the Annual Symposium on Computer Application in Medical Care pp. 261–265 (nov 1988), http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2245318/
15. Sweney, M.: Netflix gathers detailed viewer data to guide its search for the next hit. The Guardian, http://www.theguardian.com/media/2014/feb/23/netflix-viewer-data-house-of-cards
16. de la Vega, A., García-Saiz, D., Zorrilla, M., Sánchez, P.: Towards a DSL for Educational Data Mining. In: Sierra-Rodríguez, J.L., Leal, J.P., Simões, A. (eds.) Languages, Applications and Technologies SE - 8, Communications in Computer and Information Science, vol. 563, pp. 79–90. Springer International Publishing (2015)
17. Zorrilla, M., García-Saiz, D.: A service-oriented architecture to provide data mining services for non-expert data miners. Decision Support Systems 55(1), 399 – 411 (2013)

Static analysis of textual models

Iván Ruiz Rube, Tatiana Person, and Juan Manuel Dodero

Universidad de Cadiz,
Avenida de la Universidad de Cádiz, 10, 11510 Puerto Real (Cádiz), España
ivan.ruiz@uca.es,tatiana.personmontero@alum.uca.es,juanma.dodero@uca.es
http://www.uca.es

Abstract. Domain specific languages (DSLs) based on textual notations are useful to describe the semantics of a given problem. Software frameworks, such as Xtext, enable to easily design and develop textual DSLs. The use of interactive quality platforms for analysing source code such as SonarQube is increasing. For evaluating the quality of a program written with an Xtext-designed DSL, all the artifacts required by SonarQube to parse and query the source code must be developed, which becomes time-consuming and error-prone. A transformation tool and its application to a DSL are presented to bridge the gap between Xtext and SonarQube grammar formats by following a model-driven interoperability strategy.

Keywords: DSL, Xtext, SonarQube, Model-driven, Interoperability

1 Introduction

Domain specific languages (DSLs) are computer languages which are targeted to particular kinds of problems, rather than general purpose languages that are aimed at any kind of software problems [3]. In the recent years, various tools have arisen to easily develop DSLs. In this sense, Eclipse Modeling Project gathers most of the libraries, frameworks and tools to deal with the design and development of external DSLs. These tools fall into the Model Driven Software Engineering approach, which promotes model design, development, transformation and use to conduct the software process lifecycle [2]. Xtext [1] is the Eclipse Modeling Project framework used to develop programming languages and DSLs by means of a dedicated grammar language.

The use of interactive quality platforms for analysing source code such as SonarQube is increasing. SonarQube provides support for more than 20 different languages and allows users to add their own rules and advanced metrics. Providing Xtext-designed DSLs with features for evaluating the quality of the programs written with these languages could be a additional factor to contribute to the success of their adoption [4].

2 Bridging grammar formats

SonarQube includes an extension mechanism for recognising new languages and including new rules to check programs written with some of the built-in lan-

guages or the new ones. In order to evaluate the quality of programs written with Xtext-designed DSLs or to compute some metrics, we would have to develop all the artifacts required by SonarQube to parse and query the Abstract Syntax Tree (AST) of the source code parts. This task is time-consuming and error-prone, especially when it comes to maintain the consistency of the grammars while evolving the language. Afterwards, several AST visitors should be implemented to analyse quality rules or compute measures. Xtext grammars are designed by using its own specific language to describe the concrete syntax of the new language and how it is mapped to an in-memory semantic model. Xtext uses the well-known ANTLR parser [5], which implements an LL top-down parse algorithm for a subset of context-free languages. The grammar language uses Extended Backus-Naur Form (EBNF) expressions. However, the development of language recognisers in SonarQube is quite diferent. Instead of writing a grammar by using EBNF expressions, SonarQube parsing is implemented as Java classes that use a specific library called SonarSource Language Recogniser[1] (SSLR). SSLR is a lightweight Java library that provides everything required to create lexers and parsers for analysing a piece of source code.

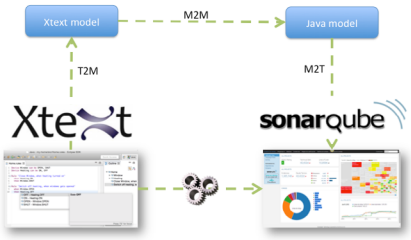


Fig. 1: Interoperability strategy

Is it possible to bridge the gap between the formats of Xtext and SonarQube grammars? This kind of problem can be tackled by adopting a model-driven interoperability approach. SonarQube describes grammars in a declarative form, whereas Xtext requires grammars written in imperative Java code. Figure 1 shows the strategy for bridging the grammar formats of both tools. This strategy consists of three phases. First, a Text to Model (T2M) transformation is carried out to obtain a Xtext model from the grammar syntax of a language written with Xtext. Second, a Model to Model (M2M) process to transform the Xtext grammar model elements into those of a Java model, according to the Sonar grammar. Finally, a Model To Text (M2T) process serializes the model from the

[1] `http://docs.sonarqube.org/display/DEV/SSLR`

previous step as the syntax required by SonarQube. In Figure 2, we can observe the abstract mapping between the elements of the grammars in both tools.

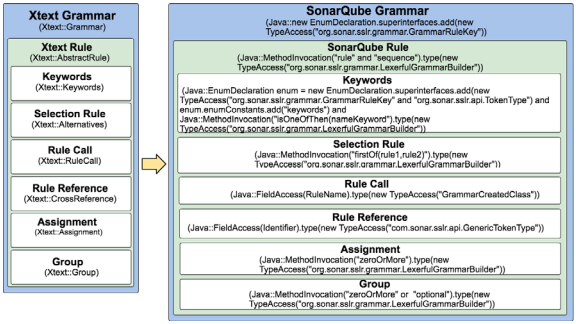


Fig. 2: Interoperability strategy

Regarding to the implementation details, the first T2M process is not necessary to be implemented from scratch, because it is already provided by Xtext. The user has to configure a specific property in the Modeling Workflow Engine (.mew) file that comes with every Xtext project. This way, when the user runs the process (represented by that workflow), an XMI version of the grammar description is automatically generated. For the sake of simplicity, the second and third transformation processes have been implemented as a single step. This prevents the need for using a model transformation engine, such as ATL, and a subsequent Java serialiser. In this vein, an Acceleo module and a set of code templates were developed to generate the Java source code artifacts required by SonarQube to recognise new languages. In addition, all the boilerplate code required by SonarQube is automatically generated. From the user's perspective, two Eclipse IDE plug-ins have been developed and made available at `https://github.com/TatyPerson/Xtext2Sonar`. These plug-ins extend the options available in the contextual menu associated to the .xtext grammar files, by adding an option to generate the source code of a Java project. The Java project has to be later imported into an existing SonarQube installation in order to support the new language.

The proposal described in this work can be also used for general purpose languages, as long as they have been developed with Xtext. In this way, the strategy was applied in Vary[2], an environment for writing and running pseudocode algorithms. A snippet of the Xtext grammar of the Vary language and its equivalent

[2] `http://tatyperson.github.io/Vary/`

4 Static analysis of textual models

for SonarQube, along with their relationships, is shown in Figure 3. Analyzing pseudocode algorithms is possible in this way with SonarQube, which provides with the basic metrics (e.g. lines of code, percentage of commented code, etc.) and quality checks according to several guideline rules, such as the abuse of global variables, the excessive number of lines of code, or whether the program subroutines are well-documented, among others.

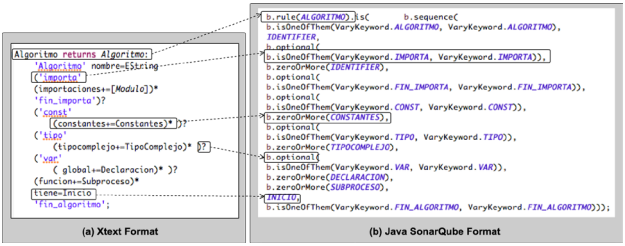


Fig. 3: Grammar code snippets

3 Conclusions

This research intends to improve the usability of the textual DSLs, by offering a tool to automatically build language recognisers for source code quality analysis platforms. The development of the tool is based on a model driven interoperability strategy that transforms Xtext grammar files into Java plug-ins for SonarQube. As a future research, a wider evaluation of this proposal on existing DSLs will be performed.

References

1. Bettini, L.: Implementing Domain-Specific Languages with Xtext and Xtend. Packt Publishing Ltd (2013)
2. Brambilla, M., Cabot, J., Wimmer, M.: Model-driven software engineering in practice. Synthesis Lectures on Software Engineering 1(1), 1–182 (2012)
3. Fowler, M.: Domain-specific languages. Pearson Education (2010)
4. Hermans, F., Pinzger, M., Deursen, A.: Domain-specific languages in practice: A user study on the success factors. In: Model Driven Engineering Languages and Systems: 12th International Conference. pp. 423–437. Springer, Berlin (2009)
5. Parr, T., Fisher, K.: LL (*): the foundation of the antlr parser generator. In: ACM SIGPLAN Notices. vol. 46, pp. 425–436. ACM (2011)

Una Propuesta de Editor Gráfico para el Modelado y la Generación de Código de Patrones de Eventos sobre Drones

Juan Boubeta-Puig[1], Juan Hernández[2], Enrique Moguel[2],
Juan Carlos Preciado[2], Fernando Sánchez-Figueroa[2]

[1] Dpto. de Ingeniería Informática, Universidad de Cádiz
juan.boubeta@uca.es
[2] Dpto. de Ingeniería de Sistemas Informáticos y Telemáticos, Universidad de Extremadura
{juanher,enrique,jcpreciado,fernando}@unex.es

Resumen. Los lenguajes de procesamiento de eventos (EPL) permiten declarar e implementar patrones de eventos que son procesados posteriormente por motores de procesamiento de eventos complejos (CEP) y así poder detectar situaciones de interés del usuario en tiempo real. Para llevar a cabo esta tarea, el usuario debe tener un alto grado de experiencia en estos lenguajes. Sin embargo, y en el ámbito de los drones, los usuarios suelen tener un vasto conocimiento en el dominio para el que se necesitan definir ciertos patrones de eventos (motores, dispositivos de navegación, pilotos automáticos, etc.), pero que son inexpertos tanto en EPLs como en el lenguaje requerido para implementar las acciones a llevar a cabo en el dron tras la detección de los eventos. En este artículo presentamos un editor de modelado de patrones con el propósito de facilitar a los usuarios finales un entorno amigable e intuitivo con el que poder definir gráficamente las situaciones críticas y relevantes que se requieran detectar en los drones, y sin necesidad de conocer ningún lenguaje de programación en particular. Además, este editor transforma estos modelos gráficos de patrones al código que los implementa.

Keywords: CEP, EPL, MDD, Editor de Modelado Gráfico, Dron.

1 Introducción y Motivación

El procesamiento de eventos complejos (*Complex Event Processing*, CEP) [1] es una tecnología emergente que permite procesar, analizar y correlacionar ingentes volúmenes de datos con el fin de detectar en tiempo real situaciones críticas o relevantes para un dominio en particular.

Los lenguajes de procesamiento de eventos (*Event Processing Languages*, EPL) permiten declarar e implementar los patrones de eventos que son procesados posteriormente por motores CEP y así poder detectar situaciones de interés del usuario en tiempo real. Para llevar a cabo esta tarea, el usuario debe tener un alto grado de experiencia en estos lenguajes. Sin embargo, los usuarios suelen tener un vasto conoci-

miento en el dominio para el que se necesitan definir ciertos patrones de eventos, pero que son inexpertos tanto en EPLs como en el lenguaje requerido para implementar las acciones a llevar a cabo tras la detección de los eventos.

Un estudio de la empresa ebizQ [2] concluye que un 84% de los encuestados consideran que la definición de estos patrones debe realizarse por expertos en el dominio, que son los que realmente disponen de todo el conocimiento necesario para ello, frente a un 16% que prefieren que sean programadores. El estudio realizado al amparo de la tesis doctoral de Boubeta-Puig [3] también corrobora esta necesidad.

Dada la cantidad de EPLs existentes, Boubeta-Puig et al. han propuesto un lenguaje de modelado específico de dominio (*Domain-Specific Modeling Language*, DSML), denominado ModeL4CEP [4], con el fin de definir un lenguaje común para que cualquier usuario pueda describir fácilmente un patrón, independientemente de su implementación. Gracias a las técnicas de desarrollo de software dirigido por modelos, este patrón definido como modelo podrá transformarse posteriormente a código de distintos EPLs con las consecuentes ventajas: oculta los aspectos técnicos de los EPLs a los usuarios finales, y la productividad mejora puesto que los modelos son más fáciles de mantener y el código generado automáticamente estará libre de errores.

Además, con objeto de acercar esta tecnología a cualquier usuario, eliminando las barreras existentes en cuanto a la programación requerida, Boubeta-Puig et al. proponen MEdit4CEP [5], una solución compuesta principalmente por un enfoque dirigido por modelos [6] y un editor gráfico e intuitivo para el modelado de patrones de eventos, así como su transformación automática tanto al código EPL a desplegar en el motor CEP, como al código de las acciones a ejecutar tras la detección de los mismos.

Una de las principales ventajas de este editor es que puede reconfigurarse mediante un modelo gráfico de dominio CEP –definido por un conjunto de tipos de eventos con sus propiedades, conforme al DSML de dominios CEP denominado ModeL4CEP. Así, el usuario dispondrá en todo momento de una interfaz gráfica común adaptada al contexto específico para el que desee definir los patrones, modificándose dinámicamente la paleta de herramientas dependiendo del dominio en cuestión. Sin embargo, las únicas herramientas sobre acciones a ejecutar tras la detección de patrones, actualmente proporcionadas por el editor, son el envío de eventos por *email* o Twitter.

En este artículo presentamos una extensión de MEdit4CEP para facilitar a los usuarios finales un entorno amigable e intuitivo con el que poder definir gráficamente las situaciones críticas y relevantes que se requieran detectar en los drones [7], y sin necesidad de conocer ningún lenguaje de programación en particular. Además, este editor transforma estos modelos de patrones al código que los implementa.

El resto del artículo se estructura de la siguiente forma. En la Sección 2 se propone el editor de modelado gráfico de patrones de eventos sobre drones y en la Sección 3 se presentan las conclusiones y el trabajo futuro.

2 Editor Gráfico Propuesto

En esta sección, se describe brevemente cómo se ha llevado a cabo la extensión de MEdit4CEP, implementado con *Graphical Modeling Framework* (GMF) y el proyec-

to Epsilon [8], para hacer posible el modelado y la generación de código de patrones de eventos sobre situaciones a detectar en los drones.

Por un lado, se ha extendido el metamodelo de patrones de eventos ModeL4CEP, especializando la metaclase *Actions* en las metaclases *ReturnToHome*, *Land* y *UAVConfiguration*. Esto soporta la definición de patrones asociados a acciones específicas a ejecutar en el propio dron: el regreso a casa (*ReturnToHome*) o el aterrizaje (*Land*); aparte de las acciones ya contempladas (el envío de eventos a una cuenta de correo o Twitter). Cabe destacar que la metaclase *UAVConfiguration* tiene dos atributos: *port* y *mode*, para indicar el número de puerto y el modo de conexión (telemetría o inalámbrico) a establecer con el dron. Además, se ha modificado la paleta de herramientas del editor, haciendo uso de *Epsilon Object Language* (EOL), para añadir dichas acciones específicas de drones como herramientas en la categoría *Actions*.

Por otro lado, se han creado nuevas reglas de validación en *Epsilon Validation Language* (EVL) que permiten validar si el patrón de eventos modelado, concretamente las acciones para drones modeladas, es conforme a su metamodelo. Por ejemplo, si el número de puerto indicado es correcto para el modo de conexión elegido.

También se han construido nuevas reglas en *Epsilon Generation Language* (EGL) para la transformación de las acciones de drones modeladas al código Python, que deberá desplegarse en el dron en cuestión para ser ejecutadas. El dron utilizado tiene un piloto automático ArduPilotMega, ya que se basa en el protocolo abierto MAVLink. La gran mayoría de drones existentes en el mercado están basados en dicho protocolo facilitando en gran medida el desarrollo de aplicaciones como la que se presenta en este artículo.

En la Figura 1 se muestra un ejemplo de patrón de eventos, modelado con el editor propuesto, que comprueba para cada evento sonoro medido por el dron si el nivel de batería es inferior al 30% de su capacidad máxima. En caso afirmativo, se enviará la orden al dron para que regrese a casa y, además, se notificará esta alerta a los usuarios interesados a través del correo electrónico.

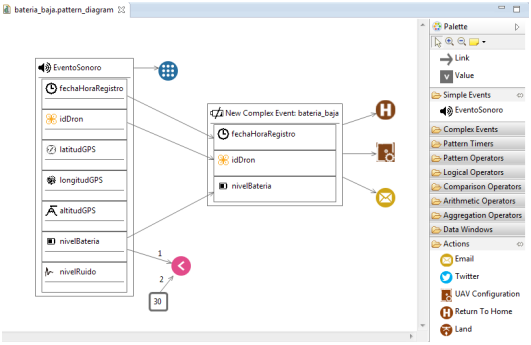


Figura 1. Patrón de eventos que detecta el nivel de batería baja en drones.

3 Conclusiones y Trabajo Futuro

En este trabajo hemos propuesto una extensión del editor gráfico de modelado MEdit4CEP, basado en ModeL4CEP, para unificar la descripción de patrones de eventos en el ámbito de los drones, representándolos como modelos; facilitando así al usuario final la definición de estos patrones sin necesidad de conocer detalles específicos sobre ningún EPL ni sobre ningún lenguaje requerido para implementar las acciones asociadas a estos patrones como, por ejemplo, el regreso del dron a casa o su aterrizaje. Además, este editor valida los patrones modelados, genera el código que los implementa, y permite importarlos y exportarlos.

Como trabajo futuro, pretendemos extender la paleta del editor gráfico, añadiendo nuevos tipos de acciones que podrán usarse durante el modelado de patrones de eventos en el ámbito de los drones, con el propósito de hacer posible la definición de situaciones críticas o relevantes que desencadenen otras acciones, aparte del regreso del dron a casa y su aterrizaje.

Agradecimientos. Este trabajo ha sido parcialmente financiado por el Ministerio de Economía y Competitividad (proyectos TIN2014-53555-REDT, TIN2015-65845-C3-3-R, TIN2015-6957-R), la Junta de Extremadura (GR15098) y los Fondos FEDER. Boubeta-Puig agradece la hospitalidad recibida durante su estancia de investigación con el Grupo Quercus de Ingeniería del Software de la Universidad de Extremadura, donde parte de este trabajo ha sido desarrollado.

Referencias

[1] D. Luckham, *Event Processing for Business: Organizing the Real-Time Enterprise*. New Jersey, USA: Wiley, 2012.
[2] BEA, "Event Processing Market Pulse 2007," ebizQ, 2007.
[3] J. Boubeta-Puig, "Desarrollo Dirigido por Modelos de Interfaces Específicas de Dominio para el Procesamiento de Eventos Complejos en Arquitecturas Orientadas a Servicios," Tesis Doctoral. Universidad de Cádiz, Cádiz, España, 2014.
[4] J. Boubeta-Puig, G. Ortiz, e I. Medina-Bulo, "ModeL4CEP: Graphical domain-specific modeling languages for CEP domains and event patterns," *Expert Systems with Applications*, vol. 42, no. 21, pp. 8095–8110, Nov. 2015.
[5] J. Boubeta-Puig, G. Ortiz, e I. Medina-Bulo, "MEdit4CEP: A model-driven solution for real-time decision making in SOA 2.0," *Knowledge-Based Systems*, vol. 89, pp. 97–112, Nov. 2015.
[6] J. Boubeta-Puig, G. Ortiz, e I. Medina-Bulo, "A Model-driven Approach for Facilitating User-friendly Design of Complex Event Patterns," *Expert Systems with Applications*, vol. 41, no. 2, pp. 445–456, Feb. 2014.
[7] Federal Aviation Administration, "Small UAS Notice of Proposed Rulemaking (NPRM)," 2015. https://www.faa.gov/uas/nprm/ [Accedido: 23/04/2016]
[8] Eclipse Foundation, "Epsilon," 2016. http://www.eclipse.org/epsilon/ [Accedido: 23/04/2016]

Towards Distributed Model Transformations with LinTra

Loli Burgueño[1], Manuel Wimmer[2], and Antonio Vallecillo[1]

[1] Universidad de Málaga. Spain. {loli,av}@lcc.uma.es
[2] TU Wien, Austria. wimmer@big.tuwien.ac.at

Abstract. Performance and scalability of model transformations are becoming prominent topics in Model-Driven Engineering. In previous works we introduced LinTra, a platform for executing model transformations in parallel. LinTra is based on the Linda model of a coordination language and is intended to be used as a middleware where high-level model transformation languages are compiled. In this paper we present the initial results of our analyses on the scalability of out-place model-to-model transformation executions in LinTra when the models and the processing elements are distributed over a set of machines.

1 Introduction

The performance and scalability of model transformations is gaining interest as industry is progressively adopting model-driven techniques, multicore computers are becoming commonplace and the cloud is extensively used. However, existing model transformation engines are mostly based on sequential and in-memory execution strategies, and thus their capabilities to transform large models in parallel and distributed environments are limited.

In previous works we introduced LinTra [2,3], a platform for executing model transformations in parallel. LinTra is based on the Linda model of a coordination language and is intended to be used as a middleware where high-level model transformation languages are compiled. Currently LinTra outperforms existing sequential and parallel model transformation engines, but we also wanted to study how the distribution aspects affect the performance of LinTra. In this paper we present the initial results of our analyses on the scalability of out-place model-to-model transformation executions in LinTra when the models and the processing elements are distributed over a set of machines.

This paper is organized as follows. After this introduction, Sect. 2 briefly describes the background of our proposal and the running example we use in the paper to illustrate our approach. Then, Sect. 3 describes an implementation and evaluation based on the case study and, finally, Sect. 4 draws some conclusions and oulines future work.

2 Backgound

2.1 LinTra

LinTra [2, 3] is a framework for the parallel execution of model transformations. It uses data-parallelism and the Blackboard paradigm [4] to store the input and output

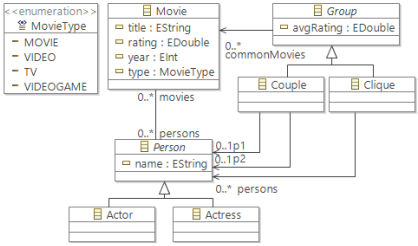


Fig. 1. IMDb Metamodel taken from [5].

models, as well as the required data to keep track of the MT execution that coordinates the agents that are involved in the transformation process. One of the key aspects of LinTra is the model and metamodel representation, wherein every entity is assigned an identifier, which is used to reference objects and to represent relationships between them. Relationships between entities are represented by storing in the source entity the identifier(s) of its target entity(ies).

In LinTra, traceability is implemented implicitly using a bidirectional function that receives as a parameter the entity identifier of the input model and returns the identifier of the output entity, regardless whether the output entities have already been created or not. This means that LinTra does not store information about traces explicitly; thus, the performance is not affected by the search for trace information.

In order to carry out the transformation process in parallel, LinTra uses data-parallelism and the Master-Slave design pattern [4]. The master's job is to launch slaves and coordinate their work. Each slave is in charge of applying the transformation to submodels of the input model (i.e., partitions) as if each partition is a complete and independent model. Since in LinTra's out-place mode the complete input model is always available, in case the slaves have data dependencies with elements that are not in the submodels they were assigned, they only have to query the Blackboard to retrieve them.

2.2 Running example

The example that we have selected uses the "Movie Database" (IMDb) proposed in the Transformation Tool Contest (TTC) 2014 [5], whose metamodel is shown in Figure 1. For exercising the distribution aspects we simply use the identity transformation, which permits checking how fast the complete model graph can be traversed and copied, i.e., focusing on the communication aspects more than on the computational aspects of the transformation itself.

3 Implementation and Evaluation

The goal is to be able to know how the distribution of data and processes affects the performance of the model transformation in LinTra. Instead of using one parallel machine,

what happens when the model is distributed among several machines, or the slaves are executed in different nodes?

In general, the analysis of the effect of data and processes distribution on the performance of a model transformation is a complex task, and a comprehensive study should deserve its own line of research. However, to initially answer this research question we have conducted some experiments in order to estimate some limits of the execution times of a model transformation when either the model, the slaves, or both, are distributed among several machines. For that we used two machines (A and B), with 4 cores each, connected in a LAN via TCP/IP. We used a very basic communication platform to implement the distributed blackboard, based on Java HashMaps to store the models and sockets for communicating the machines—hence avoiding the layers that any technological solution such as LevelDB or Cassandra would add. Our goal was to obtain the lower bounds for the execution times, independently from any solution.

We selected the IMDb-Identity model transformation case study[3], which basically traverses a 5 million elements model and produces a copy of it as the target model. We considered five configurations:

- **Config.0**: One local machine (A). All slaves are executed locally. Both source and target models are stored locally.
- **Config.1**: Two machines (A and B). All slaves are executed in A. Source and target models are stored (mingled) in both machines: elements with even identifiers in one and with odd identifiers in the other, in order to exacerbate data distribution.
- **Config.2**: Two machines (A and B). All slaves are executed in A. Source model stored in A and target model in B (best case for distribution).
- **Config.3**: Two machines (A and B). Half of the slaves are executed in each machine. Source and target models are stored both machines, mingled as in *Config.1*.
- **Config.4**: Two machines (A and B). Half of the slaves are executed in each machine. Source and target models are stored in both machines, but taking care that each slave only handles data stores locally.

For each configuration we executed the transformation on subsets of the complete model of increasing size, emulating different sizes of the database model to check how different model transformation engines scale up. The resulting execution times are shown in the left table of Fig. 2. The right table shows the speed-ups when compared with Config.0.

It is interesting to observe how data distribution has a significant impact on the model transformation execution performance. Resulting times for Config.1 and Config.3, in which the data is evenly spread and forces all slaves to access half of the data in each machine, suffer from heavy network access. Config.2 and Config.4, on the contrary, offer good results. Of course, when all slaves are executed in one machine and the target model resides remotely, the network access introduces some delay (an average speed-up of 3.8 according to our results). Config.4, in turn, provides the best case because it gets the two machines working in parallel and handling just local data (apart from the access to the shared area to gets the jobs, which resides only in machine A). The average speed-up of 0.39 is a significant achievement for this best case.

[3] http://atenea.lcc.uma.es/index.php/Main_Page/Resources/LinTra#IMDb

Execution times (seconds)					
# Elems	Config.0	Config.1	Config.2	Config.3	Config.4
200	0.05	6.65	0.03	11.85	0.01
2,000	0.04	26.85	0.09	43.24	0.02
20,000	0.11	292.84	0.39	422.51	0.03
100,000	0.84	1,602.08	3.02	3,343.25	0.09
200,000	1.53	3,087.76	6.20	3,781.01	0.74
300,000	2.14	5,364.64	12.24	3,460.83	1.01
400,000	3.00	7,230.36	13.29	7,246.77	1.44
1,000,000	16.50	24,855.59	40.02	27,726.10	3.99
2,000,000	18.38	49,711.19	90.97	47,134.36	11.53
5,000,000	52.97	104,393.49	314.42	113,122.47	26.38

Speed-up					
# Elems	Config.0	Config.1	Config.2	Config.3	Config.4
200	1.00	141.43	0.68	252.13	0.30
2,000	1.00	745.69	2.61	1,201.00	0.42
20,000	1.00	2,762.65	3.69	3,985.91	0.30
100,000	1.00	1,911.79	3.60	3,989.55	0.11
200,000	1.00	2,020.79	4.06	2,474.48	0.48
300,000	1.00	2,511.53	5.73	1,620.24	0.47
400,000	1.00	2,410.92	4.43	2,416.39	0.48
1,000,000	1.00	1,506.03	2.43	1,679.96	0.24
2,000,000	1.00	2,704.93	4.95	2,564.72	0.63
5,000,000	1.00	1,970.95	5.94	2,135.76	0.50
Average	***1.00***	***1,868.67***	***3.81***	***2,232.01***	***0.39***

Fig. 2. Execution times (left) and relative speedups (right) of distributed configurations.

Data and process distribution may have a significant impact on the performance of a model transformation. As expected, the results are very sensitive to the way in which data and processes are distributed among the nodes. We have run some experiments to show some initial results, but this issue deserves further investigations and more detailed analysis to provide more generalized results.

4 Conclusions and Future Work

This position paper has presented some initial experiments that try to estimate the effect of both data and process distribution on the performance of LinTra model transformations. The study conducted here is specific for that solution. We plan to compare these results with the emerging MT engines that also provide distribution (e.g. [1]). We also want to use different technological platforms that offer data distribution to evaluate their performance when connected with LinTra. More precisely, we want to check Infinispan with LevelBD as persistence layer whose results with LinTra are better than other datagrids when executed on a single machine [3], and the combination of Apache Spark and Cassandra. Finally, we also want to use UDP instead of TCP to see the speed-up obtained by simplifying the communication protocol used.

References

1. Benelallam, A., Gómez, A., Tisi, M., Cabot, J.: Distributed model-to-model transformation with ATL on MapReduce. In: Proc. of SLE 2015. pp. 37–48. ACM (2015)
2. Burgueño, L.: On the Quality Properties of Model Transformations: Performance and Correctness. Ph.D. thesis, Universidad de Málaga (April 2016)
3. Burgueño, L., Wimmer, M., Vallecillo, A.: A Linda-based platform for the parallel execution of out-place model transformations. Information & Software Technology. To appear
4. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns. Wiley, Chichester, UK (1996)
5. Horn, T., Krause, C., Tichy, M.: The TTC 2014 Movie Database Case. In: Proc. of the 7th Transformation Tool Contest (TTC 2014). vol. 1035, pp. 93–97. CEUR Workshops (2014)

Towards the Automation of Metamorphic Testing in Model Transformations

Javier Troya, Sergio Segura, and Antonio Ruiz-Cortés

Department of Computer Languages and Systems
Universidad de Sevilla, Spain
{jtroya, sergiosegura, aruiz}@us.es

Abstract. Model transformations are the cornerstone of Model-Driven Engineering, and provide the essential mechanisms for manipulating and transforming models. Checking whether the output of a model transformation is correct is a manual and error-prone task, this is referred to as the oracle problem in the software testing literature. The correctness of the model transformation program is crucial for the proper generation of its output, so it should be tested. Metamorphic testing is a testing technique to alleviate the oracle problem consisting on exploiting the relations between different inputs and outputs of the program under test, so-called metamorphic relations. In this paper we give an insight into our approach to generically define metamorphic relations for model transformations, which can be automatically instantiated given any specific model transformation.

Keywords: Metamorphic Testing, Model Transformation, Automation, Generic

1 Introduction

Model Transformations (MTs) are the cornerstone of Model-Driven Engineering (MDE). They provide the essential mechanisms for manipulating and transforming models. Checking whether the output of a model transformation is correct is a manual and error-prone task, this is refereed to as the oracle problem in the software testing literature. Indeed, the quality of the generated software artifacts is highly affected by the correctness of the developed model transformations. For this reason, several approaches have been proposed that verify the correct behavior of the transformations using formal methods [6,1] or certify their behavior for a selected set of test models mainly to identity bugs in a cost-effective way [2,7].

Metamorphic Testing (MT') [5] is a methodology designed to alleviate the oracle problem. Different from conventional testing strategies, MT' consists on exploiting the relations between different inputs and outputs of the program under test, so-called Metamorphic Relations (MRs). In practice, MRs define possible modifications to a test input and how those changes are propagated to the program output. A basic example of MR can be defined for the program that computes the sine function. Let us suppose we want to know the exact value of $\sin(5)$. Is an observed output of 0.091 correct? A mathematical property of the sine function states that $\sin(x) = \sin(\pi - x)$, and we can use this to test whether $\sin(5) = \sin(\pi - 5)$ without knowing the concrete values of either sine calculation. Currently, the biggest limitation of MT' has to do with the

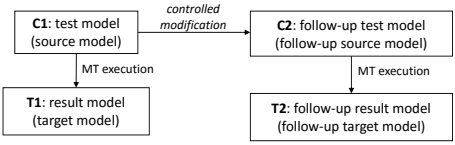


Fig. 1: Metamorphic Testing in Model Transformations.

definition of the MRs. In fact, the automatic generation of MRs has been acknowledged as one of the big challenges in MT' [5].

Figure 1 displays the scenario of MT' in MTs. We have a test model (C1), which is typically the source model. If we apply the MT, we obtain the result model (T1), i.e., the target model. By applying a controlled modification in the test model, we obtain the follow-up test model (C2). If we now apply the same MT to C2, we get the follow-up result model (T2). In this context, a MR considers the modification done in C2 with respect to C1, and the consequences that this has in T2 with respect to T1. We show an example in next section.

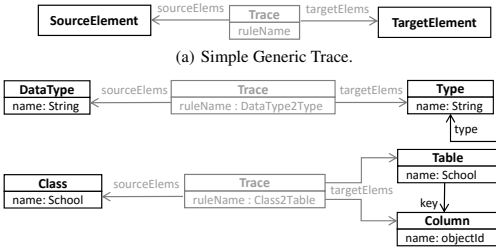
As far as we are concerned, there is only one approach that applies MT' in MTs [3]. The authors demonstrate the effectiveness and feasibility of its application, although they apply it manually in a specific scenario, for which they define the MRs. In our approach, we propose to automatically generate MRs for any model transformation, as we explain in the next section. Then, in Section 3 we describe our next steps.

2 Approach

The goal of our approach is to automate the process of metamorphic testing (MT') in model transformations (MTs). Thereby, we propose the automatic generation of metamorphic relations (MRs) for any model transformation. We work with transformations written in the ATL Transformation Language due to its importance both in academia and industry.

In order to automatically extract information out of a transformation, we make use of explicit trace models. A trace model can be automatically obtained from a transformation execution, e.g., by using Jouault's *TraceAdder* [4], and is composed of a set of traces, one for each rule execution. A trace captures the name of the applied rule and the elements of the source model (*sourceElems* relationship) that are used to create new elements in the target model (*targetElems* relationship). Therefore, by navigating the trace model, we know which target element(s) have been created from which source element(s) and by which rule. A simple example of a generic trace is shown in Figure 2(a). Please note that more than one element may appear as *sourceElems* and *targetElems*. We consider this trace as generic because each of the three elements appearing in it (*SourceElement*, *Trace* and *TargetElement*) can be instantiated in a particular scenario.

The idea of our approach is to define generic MRs for generic traces. These MRs can then be instantiated together with the generic traces. For instance, considering the generic trace of Figure 2(a), we know that if we have a test model (C1, Figure 1) and we add an element of type *SourceElement* in the follow-up test model (C2), then an



(a) Simple Generic Trace.

(b) Traces for the transformation excerpt of Listing 1.2.

Fig. 2: Generic and instantiated traces

element of type *TargetElement* is created for it in the follow-up result model (T2). This means that T2 has one more element of this type than T1. Having this information into account, we can define the first generic MR shown in Listing 1.1, written in the OCL language. Besides, the number of elements of any other type should remain the same in T1 and T2, so further MRs can be defined, such as the second one in the same listing.

Listing 1.1: Generic MRs for the addition of a *SourceElement*

```
1 T1_TargetElement.allInstances()->size()=T2_TargetElement.allInstances()->size()-1
2 T1_AnyOtherType.allInstances()->size()=T2_AnyOtherType.allInstances()->size()
```

In order to show an example of instantiation of the generic MRs shown before, we choose the well-known *Class2Relational* case study. We will focus on the excerpt of the transformation, which has been slightly modified for simplicity in the explanation, shown in Listing 1.2. If we have a source model with a *DataType* and a *Class* and we apply the transformation, the resulting trace model is depicted in Figure 2(b). We can see a trace that reflects the creation of a *Type* from a *DataType* and another one that stores the creation of a *Table* and a *Column* from a *Class*.

Listing 1.2: Excerpt of Class2Relational transformation

```
1  rule DataType2Type {
2    from
3      dt : Class!DataType
4    to
5      out : Relational!Type (
6        name <- dt.name
7      )
8  }
9
10
```

```
11 rule Class2Table {
12   from c : Class!Class
13   to
14     out : Relational!Table (
15       name <- c.name,
16       key <- key),
17     key : Relational!Column (
18       name <- 'objectId',
19       type <- thisModule.objectIdType)
20 }
```

Since these two traces are instantiations of the generic one shown in Figure 2(a), we can also instantiate the MRs shown in Listing 1.1. In particular, we have two scenarios. The first one consists of adding an element of type *DataType* in C2, what yields the

MRs shown in Listing 1.3. In the second scenario we add an element of type *Class* in C2, obtaining the MRs shown in Listing 1.4.

Listing 1.3: MRs for the addition of a *DataType* in C2

```
T1_Type.allInstances()->size()=T2_Type.allInstances()->size()-1
T1_Column.allInstances()->size()=T2_Column.allInstances()->size()
T1_Table.allInstances()->size()=T2_Table.allInstances()->size()
```

Listing 1.4: MRs for the addition of a *Class* in C2

```
T1_Column.allInstances()->size()=T2_Column.allInstances()->size()-1
T1_Table.allInstances()->size()=T2_Table.allInstances()->size()-1
T1_Type.allInstances()->size()=T2_Type.allInstances()->size()
```

3 Next Steps and Observations

In this paper we have given an insight into our approach to automate the generation of MRs for MTs. We identify generic patterns in the traces, from which we define generic MRs organized as well in patterns. For instance, one pattern is the generic trace and MRs we have shown in this paper. Despite its simplicity, we are performing ongoing works defining more patterns where elements, attributes and relationships are taken into account, so that we end up with a large set of MRs. One of the purposes of the generated MRs is to automate regression tests, since they can be checked as to whether they hold in different versions of the model transformation program and for any test model.

As mentioned, our approach takes as input one or more executions of a MT, i.e., the resulting trace models. The number of executions of the MT received and their size influence the completeness of the MRs generated. For instance, if a rule is never applied in any of the executions received as input, no MRs will consider its behavior.

Acknowledgments. This work has been partially funded by the European Commission (FEDER) and Spanish Gov. under CICYT project BELI (TIN2015-70560-R), and by the Andalusian Gov. projects THEOS (TIC-5906) and COPAS (P12- TIC-1867).

References

1. Amrani, M., Lucio, L., Selim, G.M.K., Combemale, B., Dingel, J., Vangheluwe, H., Traon, Y.L., Cordy, J.R.: A tridimensional approach for studying the formal verification of model transformations. In: Proc. of ICST. pp. 921–928. IEEE (2012)
2. Gogolla, M., Vallecillo, A.: *Tract*able Model Transformation Testing. In: ECMFA. LNCS, vol. 6698, pp. 221–235. Springer (2011)
3. Jiang, M., Chen, T.Y., Kuo, F., Zhou, Z., Ding, Z.: Testing Model Transformation Programs using Metamorphic Testing. In: SEKE'14. pp. 94–99 (2014)
4. Jouault, F.: Loosely Coupled Traceability for ATL. In: Workshop Proc. of ECMDA (2005)
5. Segura, S., Fraser, G., Sanchez, A., Ruiz-Cortes, A.: A survey on metamorphic testing. IEEE Transactions on Software Engineering (99), 1–1 (2016)
6. Troya, J., Vallecillo, A.: A Rewriting Logic Semantics for ATL. Journal of Object Technology 10, 5:1–29 (2011)
7. Vallecillo, A., Gogolla, M., Burgueño, L., Wimmer, M., Hamann, L.: Formal specification and testing of model transformations. In: Proc. of SFM. LNCS, vol. 7320, pp. 399–437. Springer (2012)

Bringing together existing Business Modeling flavors

Juan M. Vara, Valeria De Castro, David Granada, Esperanza Marcos

Grupo de Investigación Kybele, Universidad Rey Juan Carlos
Calle Tulipán S/N, 28933 Móstoles, Madrid, España

{juanmanuel.vara, valeria.decastro, david.granada, esperanza.marcos}@urjc.es

Abstract. There are currently several techniques or notations for business modeling that allow the idea of business to be explored in greater or less detail, while simultaneously helping to understand, conceptualize and represent the services that add value to an organization. All of these techniques have similarities and differences, but are in many cases complementary. However, there is no integrated environment that makes it possible to work with several models simultaneously, and much less that provides support as regards identifying, registering and managing the relationships among them. This work is a first step towards attempting to fill this lack by constructing a technological environment that will integrate tools in order to support different business modelling techniques and to register and manage the relationships among different models.

Keywords. Business Modelling, Model Engineering.

1 Motivation

The business model describes the bases upon which the firm creates, provides and captures value [5]. As stated by Teece [7], the definition of a business model implies identifying the way in which the company provides value to customers, attracts them so that they will pay for this value and converts that payment into profit.

Despite the interest raised by business model, the concept has been historically suffered from a very heterogeneous comprehension from the three main different perspectives of business models, namely technology oriented, strategy-oriented and organization-oriented. For instance, in the context of information technology business models have been classically identified with process models, while the organization theory oriented conceived the business model more as an abstract representation of a company's structure or architecture [10].

This heterogeneous comprehension is reflected for instance in the fact that there are currently a number of techniques or notations for business modelling that allow the idea of business to be explored in greater or less detail, while simultaneously helping to understand, conceptualise and represent the services that add value to an organisation.

Of these, it is possible to mention models such as the Business Canvas model [5], the Value model [4], the Service Blueprint model [1] and those models that are more process oriented, such as the PCN diagram [6] or BPMN [8].

All of these techniques have similarities and differences, but are in many cases complementary. It is therefore possible to identify relationships among them. For example, all the techniques clearly permit the identification of who are the consumers of a service, or which entities participate in a process.

Note the importance of identifying and registering these relationships, since different organisation that collaborate in a business, and even different teams within a same organisation, may use one or several of these techniques to represent the business model, with the resulting understanding problems that this could imply.

Various tools supporting some of these techniques are currently available, and basically provide the technological support needed for the creation of graphic models. However, there is no integrated environment that makes it possible to work with several models simultaneously, and much less that provides support as regards identifying, registering and managing the relationships among them.

At this point, we consider that Model Driven Engineering (MDE) (and even more specifically model engineering) can be put to work to address this scenario. In this sense, we are aligned with the already expressed thought that MDE has achieved certain levels of maturity but needs from a more applied or realistic point of view [8].

One possible solution is to broaden its scope of application, by moving to other fields where both modelling and automation can decisively help to address their problems of interest. We firmly believe this is the case with the people from the business field. We, MDE practitioners can help them to materialize and drive forward their ideas by providing them with the appropriate tooling and alleviating the inherent complexity they face when trying to bundle different pieces or concerns together, without a well-defined process or methodological basis.

All this given, this work is a first step towards attempting to fill the lack of proper support for bridging existing business modelling notations by constructing a technological environment that will integrate tools in order to support different business modelling techniques and to register and manage the relationships among different business models. Current prototype in particular presents a modelling environment for the edition of Business Canvas models, Value models and Service Blueprint models. In the medium term, we intend to automate the identification of the relationships between these and other existing models, and integrate support for another series of functionalities, such as gap analysis, in the context of business modelling.

2 Current state

INNoVaServ is a toolkit developed atop of Eclipse's EMF/GMF to create, define design and bridge different Business Models and Service Operations at different levels of abstraction and also from different points of view.

In its current shape[1], it bundles a series of DSLs supporting three different Business Modeling notations: namely the Value model, the Canvas model and the Service Blueprint model.

To illustrate the functionality delivered as of today by these tools, Fig 1, Fig 2 and Fig 3 show three different screen captures, one from the editor developed for each DSL, for the very same case study: the BicyMad business model.

Basically, BicyMad is a small family business that has been always devoted to bike selling and has decided to grow and innovate by offering incorporating the renting of electric bikes. They are consequently servitizing their business: moving from a product-based offering to a combined product-service one.

Despite the insights of this particular case, our experience while developing the tools and using them to model some business scenarios have revealed that existing notations for business modeling and service design are not only interrelated but frequently complementary.

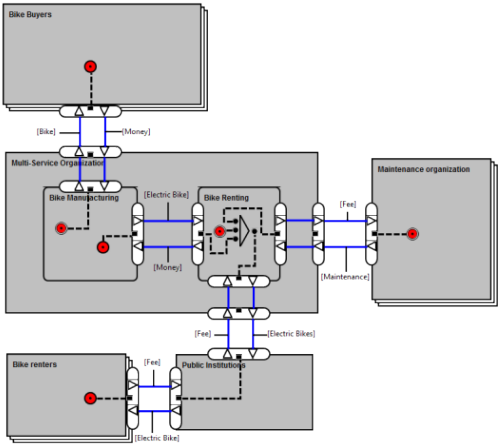


Fig 1. BicyMad e3 Value model

The e3 Value model in fact allows us to represent the stakeholders involved in a given business scenario; the underlying services composition and the value exchanges between the different stakeholders involved. However, it does not support the modelling of the insights of the different services involved in the business scenario.

By contrast, the Service Blueprint shown in Fig 2 focuses in the interactions that hold between the different stakeholders for a particular business service to take place

[1] The Eclipse pug-in can be downoad from http://www.kybele.etsii.urjc.es/innovaserv/

(in particular the figure focuses on the bike selling service). On the other hand, it does not reflect the decisions or actions triggering each step of the underlying process.

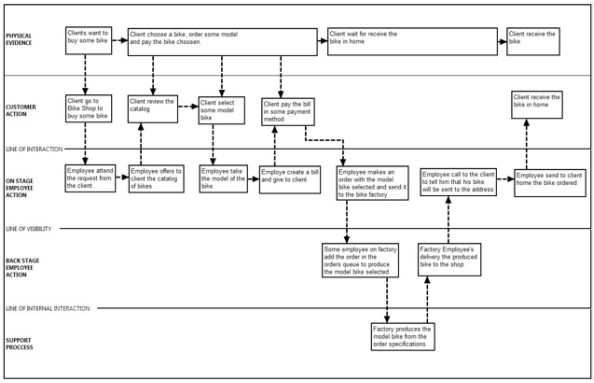


Fig 2. BicyMad Service Blueprint model

Finally, the Canvas model, depicted in Fig 3 collects general information of a given business, but it is not appropriate to capture the relationships that hold around the service.

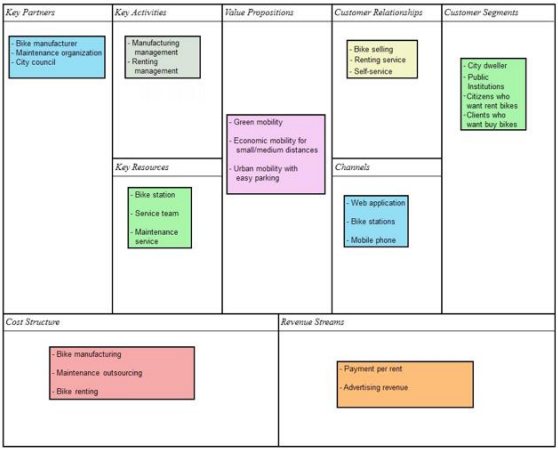


Fig 3. BicyMad Canvas model

To sum up, one model focus on the value interchanges, the other one on the underlying processes that take place and the last one in providing a very high level overview of the main factors and forces behind the business scenario.

We believe that similar findings will be bring to light as new notations are incorporated and used to model different scenarios.

3 Ongoing work

We are currently able to provide users with a unique tool that allows then to depict their business models using different notations. In addition, they are already able to persist them in some interchangeable format and we are providing them with model management capabilities in the shadow. Even more relevant is the fact that all this support comes encapsulated in an environment designed and conceived to be extended as needed, in affordable time and manner.

Regarding what remains to be done, the aim of this project was already introduced at the beginning of this work. However, in the following we break down the introduced idea into concrete objectives which are being already addressed in the context of the project:

- First of all, we are working to integrate support for some other notations for business modeling and service design, such as the already mentioned Process Chain Networks. (PCNs) [6]which is already supported by the most recent version of the toolkit.
- Next, technological bridges between the DSLs supporting each notation will be developed. The idea is to develop a set of functionalities to deal with these models and handle the information collected by means of model management tools. For instance, one might want to identify the common parts in a particular set of models, derive partial models from already existing ones, generate reports on some particular aspects, etc. Fig 4 illustrates this type of needs by using some models from the BicyMad case study. Note how the e3Value model in the middle can be used to derive initial drafts of the Service Blueprint models describing the underlying process of the service related with each value interchange depicted in the Value model.
- Connect business models with the data gathered from daily operations to support and enhance the extraction of relevant information with which to improve the performance and outcome of the services offered.

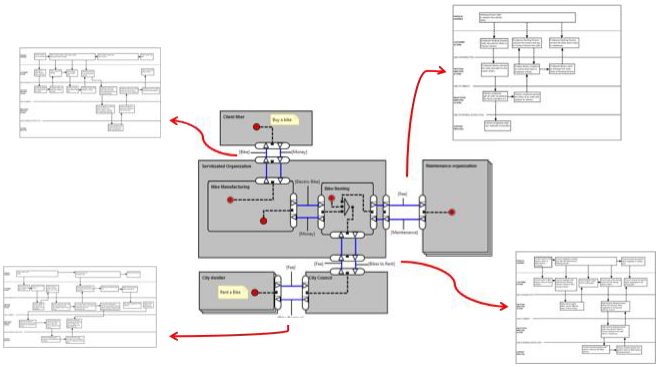


Fig 4. Service Blueprint models derived from e3Value model

Ultimately, the aim is at developing a kind of workflow to help either novel or expert users in the inception, definition and design of services using their preferred business model notation to start the process.

All in all, this project is somehow aligned with some ideas already pointed out by some renowned MDE practitioners and authors [1][3][8]: we, MDE practitioners, should try to broaden our field of application, putting modeling to work for other domains without putting the focus on technology itself but on what technology offers.

As a matter of fact, the context of this work illustrates this situation: while people from IT think probably of BPMN models or the like when talking about business modelling, people from business departments think frequently on Canvas, which provides a much more abstract and high level overview of any given business. IT people have always tried to bridge the gap between technology and business, and we thought we were doing so when using BPMN models. Unfortunately, despite some success studies and the best of our efforts, we are still far from being using the language of business people.

This is the kind of scenario that MDE practitioners might help to address, without stressing the fact that we are doing so using model-based technology, but just doing it.

Acknowledgements. This research has been partially funded by the Regional Government of Madrid under the SICOMORo-CM (S2013/ICE-3006) project, by ELASTIC (TIN2014-52938-C2-1-R) project, financed by the Spanish Ministry of Science and Innovation, and by the Service Science, Management and Engineering-GES2ME Research Excellence Group (Ref. 30VCPIGI15) co-funded by Rey Juan Carlos University and Banco Santander.

References

[1] Bézivin, J. (2012). Where will be MDE in 2030?. http://www.nii.ac.jp/userimg/lectures/20120117/Lecture4.pdf. Last accessed 20/03/2016.
[2] Bitner, M., Ostrom, A., & Morgan, F. (2008). Service Blueprinting: A Practical Technique for Service Innovation, California Management Review, vol. 50, no. 3, Spring 2008, pp 66-94.
[3] Brambilla, M. (2015) Un-Modeling. Because Modeling is dead. Panel on Software Modeling at MODELSWARD 2015, Eseo, France.
[4] Gordijn, J., &Akkermans, J.M., (2003). Value based requirements engineering: exploring innovative e-commerce idea. Requirements Engineering Journal, Vol 8 (2), 114 -134.
[5] Osterwalder, A., &Pigneur, Y. (2010). Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers. John Wiley and Sons.
[6] Sampson, S. (2012): Visualizing Service Operations. Journal of Service Research, 15 (2), 182-198.
[7] Teece, D. J. (2010). Business models, business strategy and innovation. *Long range planning*, *43*(2), 172-194.
[8] White, S. A. (2004). Introduction to BPMN. IBM Cooperation, 2(0), 0.
[9] Whittle, J., Hutchinson, J., & Rouncefield, M. (2014). The State of Practice in Model-Driven Engineering. *IEEE Software*, *31*(3), 79–85. http://doi.org/10.1109/MS.2013.65
[10] Wirtz, B. W., Pistoia, A., Ullrich, S., & Göttel, V. (2015). Business models: origin, development and future research perspectives. Long Range Planning.

Análisis de transformaciones de modelos ATL con AnATLyzer

Jesús Sánchez Cuadrado, Esther Guerra and Juan de Lara

Grupo de investigación Miso (`http://miso.es`)
Universidad Autónoma de Madrid (Spain)

Resumen Las transformaciones de modelos son un elemento clave del Desarrollo de Software Dirigido por Modelos puesto que permiten automatizar muchas tareas de manipulación de modelos. Por tanto, disponer de métodos que permitan detectar errores no triviales resulta esencial. Sin embargo no existen herramientas prácticas de análisis de transformaciones que sean capaces de tratar con transformaciones complejas.
En esta demostración se presentará anATLyzer, un analizador estático para transformaciones ATL que hace uso de *constraint solving* para mejorar la precisión del análisis. anATLyzer no se limita a un subconjunto de ATL sino que intenta cubrir ATL completamente. Se integra con el editor de ATL en Eclipse, y ofrece servicios adicionales como visualización y quick fixes, así como una API para ser utilizado de manera programática.
La demostración se ilustrará con un ejemplo sobre el que se mostrarán algunos de los tipos de errores que hemos encontrado analizando transformaciones del Zoo de ATL, con el objetivo de motivar la necesidad de este tipo de herramientas y mostrar sus características principales.

Keywords: Transformaciones de modelos, ATL, Análisis estático, Desarrollo dirigido por modelos.

1. Introducción

Las transformaciones de modelos son un elemento clave del Desarrollo de Software Dirigido por Modelos (DSDM), puesto que permiten automatizar la manipulación de los modelos. Aunque en los últimos años se han propuesto toda una serie de lenguajes de transformación [3,4,5,7], las implementaciones de estos lenguajes han prestado poca atención al análisis de las transformaciones escritas con ellos. Así, la situación actual es que muchos lenguajes populares como ATL [4] y ETL [5] no tienen ningún tipo de análisis estático previo a la ejecución, lo que lleva a costosos ciclos de ejecución-prueba para depurar la transformación. Otros lenguajes, como QVT [7], disponen de cierto chequeo de tipos pero no ofrecen ningún tipo de análisis relativo a la ejecución de reglas.

Para abordar esta problemática hemos creado anATLyzer, un analizador estático de transformaciones ATL. La siguiente sección presenta sus características principales, y la Sección 3 describe brevemente el contenido de la demostracioń que se realizará.

2. AnATLyzer

El objetivo de ANATLYZER es la detección avanzada de errores en transformaciones ATL [1]. El proceso de análisis tiene tres partes. Primero se lleva a cabo un chequeo de tipos, en el que se realiza la inferencia de tipos de las expresiones OCL y se asigna a cada nodo del árbol sintáctico un tipo. Esta etapa no es trivial ya que ATL está tipado dinámicamente y su compilador no emite errores más allá de cuestiones sintácticas. En la segunda etapa se analizan las relaciones entre reglas y se detectan problemas asociados. En estas dos primeras etapas, con el objetivo de mejorar la precisión del análisis, algunos errores se marcan como potenciales. En la tercera etapa del análisis se utiliza satisfacción de restricciones (*constraint solving*) para confirmar o descartar estos errores potenciales. Para cada error potencial se calcula su *path condition* como una expresión OCL que describe las características de aquellos modelos que harían que el flujo de la transformación llegara a la localización que contiene el problema. Esta expresión se utiliza como entrada de la herrramienta USE Validator [6], que intenta encontrar un modelo que la cumpla. Si tal modelo existe significa que el error es real ya que existe al menos una configuración que haría fallar la transformación, mientras que en caso contrario se descarta el error potencial.

Nuestra técnica de análisis amplía el rango de errores que se pueden detectar y permite un uso eficiente del *constraint solver* mediante la poda de los metamodelos de entrada (*pruning*). Como dato significativo, en nuestros experimentos con transformaciones del Zoo de ATL[1] el tiempo de ejecución del *constraint solver* es en la mayoría de los casos inferior a medio segundo. También cabe destacar que todas las transformaciones del zoo tienen al menos un problema. La Tabla 1 muestra algunos de los errores más significativos que se detectan con ANATLYZER. El analizador actualmente detecta 45 tipos distintos de error.

	Descripción	**Tipo de error**	**Precisión**
1	Propiedad u operación no encontrada	tipado	estática
2	Declaración no coincide con el tipo inferido	tipado	estática
3	Acceso a objeto sin definir (OclUndefined)	navegación	solver
4	Propiedad obligatoria no inicializada	integridad del modelo destino	estática
5	Binding resuelto por regla con tipo incompatible	integridad del modelo destino	solver
6	Binding no resuelto	reglas	solver
7	Conflicto de reglas	reglas	solver

Tabla 1: Algunos de los problemas detectados por ANATLYZER.

Entre los errores que ANATLYZER detecta se incluyen el acceso a una propiedad no definida en el meta-modelo (error #1) o si los tipos declarados en la transformación no coinciden con el tipo inferido (error #2). Este tipo de problemas están relacionados con el tipado de la transformación, no requieren el uso del *constraint solver*, y decimos que su precisión es "estática". También se detectan errores relativos a la navegación en expresiones OCL, como por ejemplo el acceso a un objeto no definido (i.e., un "null pointer exception", error #3).

[1] `http://www.eclipse.org/atl/atlTransformations/`

Este tipo de error a veces requiere confirmación mediante el uso del *constraint solver*. Una clase importante de errores son aquellos que afectan a la integridad del modelo generado, esto es, si la transformación genera siempre modelos que son conformes al meta-modelo destino. Por ejemplo, es común olvidar la inicialización de una propiedad obligatoria al crear un objeto destino (error #4). Un error más sutil, que requiere *constraint solving* para su detección, es la asignación incorrecta en una propiedad debido a que la regla que resuelve el *binding* correspondiente tiene un tipo incorrecto en su parte *to* (error #5). También se detectan situaciones en las que puede que un *binding* no sea resuelto por ninguna regla (error #6) o conflictos de reglas (i.e., reglas cuyos patrones de entrada no son exclusivos, error #7).

De cara al desarrollador de transformaciones, ANATLYZER se integra con el editor estándar de ATL en Eclipse para realizar el análisis estático al mismo tiempo que se desarrolla la transformación. La Figura 1 muestra una captura de pantalla del entorno. La etiqueta 1 muestra el editor de ATL con marcadores de error producidos por ANATLYZER. Estos marcadores también aparecen en la vista de problemas de Eclipse, y además, se ha creado una vista específica para el análisis de transformaciones ATL (etiqueta 2). La ejecución del *constraint solver* se realiza en segundo plano y se ha implementado un mecanismo para que su ejecución sea incremental, es decir, cuando se cambia la transformación sólo hay que volver a invocarlo para aquellos errores afectados por el cambio. Por otra parte, dado un error es interesante disponer de mecanismos automatizados para arreglarlo. Para esto se ha implementado un amplio catálogo de *quick fixes* [2] (etiqueta 3). Finalmente, se está trabajando en la visualización de los problemas para facilitar su comprensión (etiqueta 4).

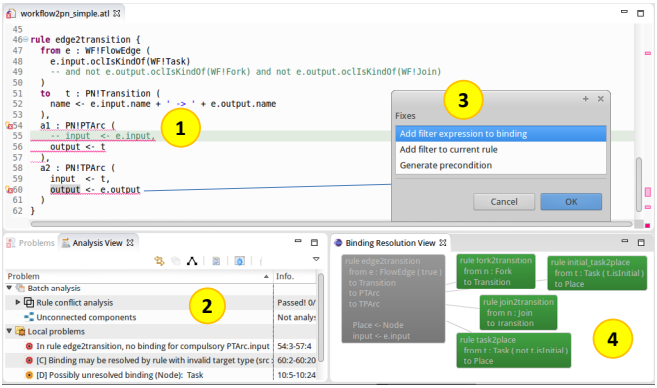


Figura 1: Entorno de desarrollo en Eclipse

Por último, anATLyzer ofrece una API que permite invocar al analizador de manera programática, inspeccionar los resultados, y obtener una versión del árbol sintáctico de ATL enriquecida con información de tipos. Creemos que esta característica es muy útil para quienes investigan en transformaciones ATL.

3. Demostración

Esta demostración pretende dar a conocer anATLyzer a la comunidad española de DSDM con tres objetivos principales. Por una parte, animar a los programadores ATL a que prueben la herramienta y tengan la oportunidad de descubrir si sus transformaciones tienen errores. También, como se ha mencionado, pensamos que anATLyzer es de utilidad para cualquier investigador interesado en el análisis de transformaciones. Por último, estamos interesados en obtener realimentación acerca de *bugs*, mejoras y sobre la utilidad de la herramienta por parte de que aquellos que la usen.

En la demostración se presentarán las características de anATLyzer, utilizando como guía ejemplos simples pero inspirados en casos reales encontrados en transformaciones del Zoo de ATL. A través de estos ejemplos se mostrarán los errores más importantes que se pueden detectar y se motivará la utilidad de la herramienta. También se mostrarán otros aspectos como su configuración, rendimiento, *quick fixes*, etc. La herramienta está disponible en `http://miso.es/tools/anATLyzer.html`, donde se incluye el código fuente, un *update site* para su instalación, *screencasts* y documentación.

Agradecimientos. Trabajo financiado por el MINECO (TIN2014-52129-R), la Comunidad de Madrid (S2013/ICE-3006) y la Comisión Europea (FP7-ICT-2013-10, #611125).

Referencias

1. J. S. Cuadrado, E. Guerra, and J. de Lara. Uncovering errors in atl model transformations using static analysis and constraint solving. In *ISSRE'14*, pages 1–11. IEEE Computer Society, 2014.
2. J. S. Cuadrado, E. Guerra, and J. de Lara. Quick fixing ATL model transformations. In *MoDELS'15*, pages 146–155. IEEE, 2015.
3. J. S. Cuadrado, J. G. Molina, and M. M. Tortosa. RubyTL: A Practical, Extensible Transformation Language. In *ECMDA-FA'06*, volume 4066 of *LNCS*, pages 158–172. Springer, 2006.
4. F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31 – 39, 2008.
5. D. S. Kolovos, R. F. Paige, and F. Polack. The epsilon transformation language. In *ICMT'08*, volume 5063 of *LNCS*, pages 46–60. Springer, 2008.
6. M. Kuhlmann, L. Hamann, and M. Gogolla. Extensive validation of OCL models by integrating SAT solving into USE. In *TOOLS (49)*, volume 6705 of *LNCS*, pages 290–306. Springer, 2011.
7. QVT. `http://www.omg.org/spec/QVT/`.

Una herramienta para evaluar el rendimiento de aplicaciones intensivas en datos*

Abel Gómez and José Merseguer

Departamento de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, Spain
{abel.gomez|jmerse}@unizar.es

Resumen Las aplicaciones intensivas en datos (AID) que usan tecnologías de *Big Data* se están convirtiendo en una parte importante del mercado de desarrollo de software. Sin embargo, las técnicas —y su automatización— para el asesoramiento de la calidad para este tipo de aplicaciones es claramente insuficiente. El proyecto DICE H2020 tiene como objetivo definir metodologías y crear herramientas para desarrollar y monitorizar AID mediante técnicas de ingeniería dirigida por modelos. En este artículo presentamos un componente clave del proyecto DICE: su herramienta de simulación. Esta herramienta es capaz de evaluar el rendimiento de AID simulando su comportamiento mediante modelos de redes de Petri. Como complemento, existe a disposición un vídeo mostrando la herramienta en http://tiny.cc/z1qzay.

Keywords: Aplicaciones Intensivas en Datos (AID), Lenguaje de Modelado Unificado (UML), Redes de Petri (RdP)

1. Introducción

La creciente disponibilidad de infraestructuras en la nube, así como clusters distribuidos o nuevos modelos de programación (como *MapReduce*), están teniendo un fuerte impacto en el mercado del desarrollo de software y en las llamadas *aplicaciones intensivas en datos* (AID). En esta nueva realidad, es necesario proporcionar metodologías que, siendo capaces de lidiar con la creciente complejidad del software y su despliegue, permitan incrementar la productividad.

El proyecto DICE [2] aspira a definir este nuevo marco de trabajo para AID que permita explotar las tecnologías de *Big Data* en nubes públicas o privadas. DICE propone técnicas, metodologías y herramientas para diseñar, desplegar, monitorizar y refactorizar AID. En lo que concierne al diseño de las aplicaciones el Lenguaje de Modelado Unificado (UML) es fundamental en DICE. Aquellas características inherentes a una AID que no están contempladas en la propia especificación de UML (e.g., tecnología de *Big Data* a utilizar) se definen usando un nuevo perfil proporcionado por DICE [4]. Un aspecto clave del proyecto DICE es su ecosistema de herramientas para simular, analizar y optimizar las

* Trabajo financiando por la UE bajo el programa H2020 (n^o 644869 – DICE), el MICINN (ref. TIN2013-46238-C4-1-R) y el Gobierno de Aragón (ref. T94 – DisCo)

aplicaciones modeladas en DICE. Una de estas herramientas es la llamada *DICE Simulation Tool*, que permite evaluar los requisitos de rendimiento de una AID usando las redes de Petri [1] (RdP). A continuación, la sección 2 introduce las bases sobre las que se construye la herramienta de simulación con un sencillo ejemplo, mientras que la sección 3 describe la arquitectura de dicha herramienta. Finalmente, la sección 4 cierra el artículo.

2. Un ejemplo ilustrativo

La figura 1a muestra un diagrama de actividades de UML sencillo. El diagrama consta de un único nodo inicial (*Inicio*), un único nodo final (*Fin*) y dos acciones que se ejecutan de forma alternativa (*A1* y *A2*). Se puede observar que se han aplicado los estereotipos *GaWorkloadEvent* y *GaStep*. Estos estereotipos, definidos en el profile MARTE [6], son importados por el perfil de DICE para especificar tanto la carga del sistema como los requisitos de demanda de servicio. La aplicación de *GaWorkloadEvent* al nodo inicial marca una carga del sistema cerrada con una población inicial de 3 trabajos, mientras que, la aplicación de *GaStep* a los flujos que parten del nodo de decisión indican que las probabilidades de que se ejecuten las actividades *A1* o *A2* son `$p1` y `$p2` respectivamente. Respecto a `$p1` y `$p2`, éstas son variables que pueden configurarse posteriormente en el momento de evaluar el sistema (sec. 3). Finalmente, el estereotipo *GaStep* aplicado a *A1* y *A2* especifica que, por término medio, la actividad *A1* requerirá un segundo de uso de CPU para su ejecución y la actividad *A2* requerirá dos.

Si bien el modelo UML es óptimo para que el ingeniero especifique tanto el *workflow* de la AID como sus características inherentes, no lo es para llevar a cabo una evaluación y asesoramiento de las propiedades cuantitativas de la AID. Es por ello que es necesario transformar el modelo UML en una RdP. Las RdP estocásticas son un formalismo gráfico que posibilita el modelado, análisis y evaluación de los sistemas. La figura 1b muestra la correspondiente RdP que permitirá evaluar el rendimiento del sistema especificado en la figura 1a. En ella, los nodos circulares representan *lugares*, los nodos rectangulares con relleno negro *transiciones inmediatas*, y los nodos rectangulares con relleno blanco *transiciones con tiempo*. Se puede observar que el lugar llamado *Inicio* contiene tres *tokens* (como indica la población inicial especificada en el modelo UML); las transiciones inmediatas t_2 y t_3 especifican un peso de `$p1` y `$p2` respectivamen-

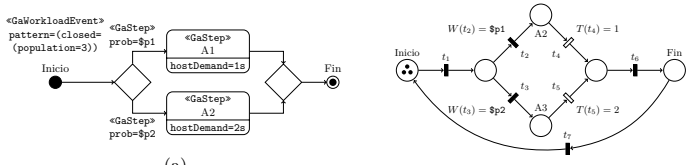


Figura 1: Ejemplo de un diagrama de actividad y su correspondiente red de Petri

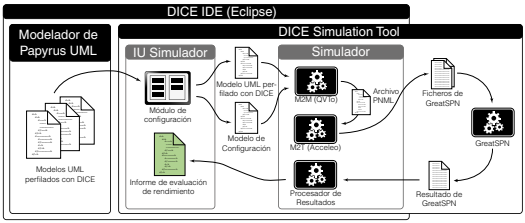


Figura 2: Arquitectura de alto nivel de la herramienta

te; y las transiciones con tiempo t_4 y t_5 especifican un tiempo medio de disparo de 1 y 2 unidades de tiempo respectivamente. El análisis de esta red de Petri permitirá razonar sobre el rendimiento teórico del sistema modelado.

3. La herramienta de simulación de DICE

La *herramienta de simulación de DICE* permite no sólo automatizar la transformación de un diagrama UML a su modelo formal, sino también controlar el proceso de simulación de forma transparente para el usuario. Además esta herramienta implementa el *Profile* de DICE y en un futuro tendrá capacidades para redefinir el diseño de la aplicación en función de las ejecuciones reales de la misma. La figura 2 muestra, de forma simplificada, su arquitectura.

El *DICE-IDE* es el entorno de desarrollo de DICE. Está construido sobre Eclipse [9] y en él se integran todos los componentes proporcionados por el proyecto. Como paso previo a una evaluación de rendimiento, se debe modelar una AID con UML y el perfil de DICE. Para realizar el modelado, DICE emplea *Papyrus UML* [8] (fig. 3 izquierda). La interfaz de usuario del simulador (*IU Simulador*) es un *plug-in* de Eclipse que se integra perfectamente en el DICE-IDE y proporciona una fachada sencilla que interactúa con las herramientas de análisis subyacentes de forma transparente. El *módulo de configuración* permite

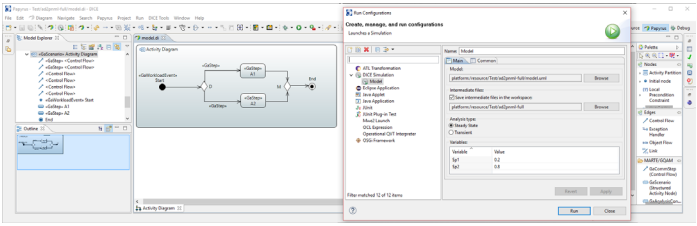


Figura 3: El entorno de desarrollo de DICE y el módulo de configuración

al ingeniero instanciar los parámetros de la AID especificados mediante el profile de DICE, como por ejemplo la carga de trabajo. Como resultado de una configuración, el *Simulador* recibe dos ficheros: el modelo perfilado con DICE (modelo a analizar) y el *Modelo de configuración*.

El *Simulador* es el componente que orquesta la interacción entre todas las herramientas involucradas. Para ello, ejecuta los siguientes pasos: (i) transforma el modelo UML perfilado en una RdP según el estándar PNML [5] empleando una transformación modelo-a-modelo (M2M); (ii) convierte la RdP estándar al formato específico de la herramienta de análisis (*GreatSPN* [3]), con una transformación modelo-a-texto (M2T); (iii) ejecuta el análisis empleando *GreatSPN*; y (iv) construye un *informe de evaluación de rendimiento* independiente de la herramienta a partir del resultado proporcionado por la herramienta de análisis. Dicho *informe de evaluación de rendimiento* se muestra al usuario en la IU del simulador empleando los conceptos definidos en el modelo UML inicial.

4. Conclusiones

En este artículo hemos presentado la *herramienta de simulación* del proyecto *DICE*. La herramienta es capaz de proporcionar un *informe de evaluación del rendimiento* de una AID a partir de un modelo UML inicial anotado con el perfil de DICE [4]. En su estado actual, el prototipo cubre todos los pasos del flujo de trabajo de la simulación. La herramienta se integra completamente en el entorno de desarrollo de DICE y proporciona una interfaz amigable que oculta al usuario los detalles de la herramienta de análisis subyacente. La herramienta (`http://tiny.cc/z1qzay`), se ha publicado bajo una licencia de código abierto [7].

Referencias

1. Ajmone-Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modeling with Generalized Stochastic Petri Nets. John Wiley and Sons (1994)
2. Casale, G., et al.: DICE: Quality-driven Development of Data-intensive Cloud Applications. In: Proceedings of the Seventh International Workshop on Modeling in Software Engineering. pp. 78–83. IEEE Press, NJ, USA (2015)
3. Dipartamento di informatica, Università di Torino: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets (2016), URL: `http://www.di.unito.it/~greatspn/index.html`
4. Gómez, A., Merseguer, J., Di Nito, E., Tamburri, D.A.: Towards a UML Profile for Data Intensive Applications. In: Proceedings of the 2nd International Workshop on Quality-aware DevOps. pp. 18–23. ACM (2016), DOI: 10.1145/2945408.2945412
5. ISO: Systems and software engineering – High-level Petri nets – Part 2: Transfer format. ISO/IEC 15909-2:2011, Geneva, Switzerland (2008)
6. OMG: UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems, Version 1.1 (June 2011)
7. The DICE Consortium: DICE Simulation Repository (2016), URL: `https://github.com/dice-project/DICE-Simulation`
8. The Eclipse Foundation: Papyrus (2016), URL: `https://eclipse.org/papyrus/`
9. The Eclipse Foundation: Website (2016), URL: `http://www.eclipse.org/`

Computing Repairs for Constraint Violations in UML/OCL Conceptual Schemas

Xavier Oriol[1], Ernest Teniente[1], and Albert Tort[2]

[1] Department of Service and Information System Engineering
Universitat Politècnica de Catalunya, Barcelona, Spain
{xoriol,teniente}@essi.upc.edu
[2] Sogeti España, Barcelona, Spain
albert.tort-pugibet@sogeti.com

Abstract. La actualización de los contenidos de una base de información puede conllevar la violación de una o más restricciones de integridad de su esquema. La forma clásica de tratar dicho problema consiste en rechazar la actualización cuando su aplicación induce dichas violaciones. En este artículo, seguimos un enfoque distinto basado en computar, automáticamente, los reparadores de una modificación. Es decir, computamos los mínimos cambios que, añadidos a los de la propia actualización pedida, llevan a la base de información a un nuevo estado donde todas las restricciones son satisfechas. Nuestro método es independiente del lenguaje usado para definir el esquema y sus restricciones puesto que se basa en una formulación lógica de ambas, aunque mostramos su utilidad con esquemas UML/OCL puesto que son de gran relevancia en la comunidad de modelización conceptual de hoy en día.
Nuestro método puede ser usado para mantener la consistencia de una base de información después de aplicar una modificación, así como para tratar el problema de reparar operaciones no ejecutables. El fragmento de OCL que se trata es expresivamente equivalente al álgebra relacional, e identificamos un subconjunto de este con propiedades beneficiosas para el proceso de reparación. La eficiencia del método ha sido probada experimentalmente.

Artículo original disponible en http://dx.doi.org/10.1016/j.datak.2015.06.006

Software Modernization Revisited: Challenges and Prospects

Hugo Bruneliere[1], Jordi Cabot[2,3], Javier Luis Cánovas Izquierdo[3], Leire Orue-Echevarria Arrieta[4], Oliver Strauss[5], and Manuel Wimmer[6]

[1] AtlanMod Team (Inria, Mines Nantes, LINA). Nantes. France
[2] ICREA. Barcelona. Spain
[3] UOC. Barcelona. Spain
[4] Competence Team Software Engineering (Fraunhofer IAO). Stuttgart. Germany
[5] ICT European Software Institute Division, (TECNALIA). Bilbao. Spain
[6] Business Informatics Group (Vienna University of Technology). Viena. Austria

Resumen El número de proyectos de modernización abordados por compañías de software crece cada día debido a cambios tecnológicos, que obligan a mejorar y evolucionar los sistemas para evitar su obsolescencia. Estos proyectos no siempre reciben la atención adecuada y se inician fundamentalmente para adecuarse a modas y no a limitaciones tecnológicas o problemas reales. Desde nuestra experiencia, muchos desarrolladores tienen una vista parcial de la complejidad y consecuencias de estos proyectos.

Un proyecto de modernización tiene normalmente tres fases. Primero la fase de ingeniería inversa estudia el estado actual del sistema software a modernizar. Muchas herramientas existentes utilizan técnicas de modelado para obtener modelos que representen el sistema a un alto nivel de abstracción. Algunos ejemplos de estos modelos son los diagramas de clase UML, máquinas de estado o flujos de trabajo. En segundo lugar, la fase de ingeniería directa analiza los modelos obtenidos y los transforma (si es necesario) para representar el sistema software modernizado. Finalmente, un grupo de desarrolladores o herramientas de generación de código (o una combinación de ambos) utilizan estos modelos para producir el código para la plataforma software destino.

Mientras que hay herramientas para facilitar el desarrollo de estas fases, no existe una metodología global que guíe a los desarrolladores durante el proceso de modernización. Además, tampoco existe soporte para evaluar el riesgo o la calidad del producto software final. Basándonos en nuestra experiencia en el uso y desarrollo de herramientas para proyectos de modernización de software, en este artículo discutimos una serie de factores importantes a considerar y presentamos algunas recomendaciones con el objetivo de maximizar el éxito de los proyectos de modernización.

(Publicado en IEEE Computer 48(8): 76-80. 2015)

Gestión de Datos

Gestión de Datos

Coordinadores: Sergio Ilarri y José Ramón Paramá

Maria Del Mar Roldán-García, Jose García-Nieto y Jose F. Aldana-Montes. *Un Repositorio RDF para la Integración de Flujos de Datos de Analítica Web y Comercio Electrónico.* (Completo)

Ana Cerdeira-Pena, Antonio Fariña, Javier D. Fernández y Miguel A. Martínez-Prieto. *v-RDFCSA: Compresión e Indexación de Colecciones de Versiones RDF.* (Completo)

José M. Giménez-García, Javier D. Fernández y Miguel A. Martínez-Prieto. *Compresión de Big Semantic Data basada en HDT y MapReduce.* (Completo)

Miguel Sánchez Cabrera, Manuel Barrena, Pablo Bustos García de Castro y Pablo García Rodríguez. *Arquitectura software basada en tecnologías Smart para agricultura de precisión.* (Completo)

Antonio Corral, Francisco García-García, Luis Iribarne y Michael Vassilakopoulos. *Distance Range Queries in SpatialHadoop.* (Completo)

Sergio Ilarri y Slavcho Ivanov. RecSim. *Hacia la Evaluación de Recomendación Utilizando un Simulador de Entornos Móviles.* (Corto)

David Álvarez, José R.R. Viqueira y Alberto Bugarín. *Aproximación a la búsqueda basada en términos sobre conjuntos de datos medioambientales.* (Corto)

Shahed Almobydeen, José R.R. Viqueira y Manuel Lama Penín. *A Federated Approach for Array and Entity Environmental Linked Data.* (Corto)

Diego Ferrón Lea, Sebastián Villarroya, José R.R. Viqueira y Tomás F. Pena. *Procesamiento paralelo de datos medioambientales con Apache Spark.* (Corto)

Xavier Oriol, Ernest Teniente y Guillem Rull. *TINTIN. Comprobación incremental de aserciones SQL.* (Demo)

Ismael Rodríguez Hernandez, Raquel Trillo-Lado y Roberto Yus. WikInfoboxer. *A Tool to Create Wikipedia Infoboxes Using DBpedia.* (Demo)

I. Navas-Delgado, M.J. García-Godoy, Esteban López-Camacho, Maciej Rybinski , Armando Reyes-Palomares, M.A Medina, José F. Aldana-Torres: *kpath: integration of metabolic pathway linked data,* Database-The Journal of Biological Databases and Curation (Oxford) vol. 2015, 1-11, 2015 (doi: 10.1093/database/bav053). (Relevante)

Miguel A. Martínez-Prieto, Nieves Brisaboa, Rodrigo Cánovas, Francisco Claude, Gonzalo Navarro: *Practical Compressed String Dictionaries*, Information Systems, 56 (2016) 73–108. (Relevante)

S. Ilarri, T. Delot y R. Trillo-Lado: *Las Redes de Vehículos desde la Perspectiva de Gestión de Datos.* IEEE Communications Surveys and Tutorials, 17(4), 2015. (DOI: 10.1109/COMST.2015.2472395). (Relevante)

María Teresa Gómez López, Rafael M. Gasca, José Miguel Pérez-Álvarez: *Validación y Diagnosis en tiempo de Ejecución sobre Reglas de Cumplimiento en Datos para Procesos de Negocio.* 48: 26-43 (2015). (Relevante)

Un Repositorio RDF para la Integración de Flujos de Datos de Analítica Web en Comercio Electrónico

Maria del Mar Roldán-García, Jose García-Nieto, and Jose F. Aldana-Montes

Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga
{mmar, jnieto, jfam}@lcc.uma.es

Resumen La Analítica Web supone hoy en día una tarea ineludible para las empresas de comercio electrónico, ya que les permite analizar el comportamiento de sus clientes. El proyecto Europeo SME-Ecompass tiene como objetivo desarrollar herramientas avanzadas de análitica web accesibles para las PYMES. Con esta motivación, proponemos un servicio de integración de datos basado en ontologías para recopilar, integrar y almacenar información de traza web procedente de distintas fuentes. Estas se consolidan en un repositorio RDF diseñado para proporcionar semántica común a los datos de análisis y dar servicio homogéneo a algoritmos de Minería de Datos. El servicio propuesto se ha validado mediante traza digital real (Google Analitics y Piwik) de 15 tiendas virtuales de diferentes sectores y países europeos (UK, España, Grecia y Alemania) durante varios meses de actividad.

Keywords: Análisis Web, Ontologías, Integración de datos, RDF y Google Analytics.

1. Introducción

El Análisis Web está tomando cada vez más relevancia dentro del comercio electrónico, ya que permite a los comerciantes y gestores de sitios web obtener información relevante sobre el comportamiento de los clientes cuando visitan sus tiendas online. Además, las aplicaciones de analítica web ofrecen una visión cualitativa y cuantitativa sobre el posicionamiento de una web en el mercado y ayudan a obtener indicios sobre el impacto de cierta oferta o campaña publicitaria. Los procesos de análisis web se basan en el trazado de la "huella electrónica o digital" del visitante del sitio online, lo cual incluye desde metadatos de geolocalización, preferencias e incluso ratio de conversión, es decir, proporción de visitantes que terminan efectuando una compra. Estos datos se contrastan con indicadores de rendimiento (Key Performance Indicators, KPIs) para mejorar la tasa de éxito global del sitio de comercio electrónico analizado.

Actualmente existen un gran número de herramientas de analítica web, algunas de ellas muy conocidas, como: Google Analytics, Piwik y Clicky. Estas

herramientas se utilizan, no solo para trazar y medir el tráfico web en un sitio específico, si no también para analizar su actividad comercial, en términos de compras, beneficios, etc. Sin embargo, estas herramientas normalmente se centran en atributos numéricos de traza (contadores) y medidas de bajo nivel, sin la posibilidad de obtener análisis más sofisticados, como la clasificación de tipología de clientes y su evolución respecto a cierto tipo de producto. En la mayoría de los casos, estos análisis están disponibles sólo para sus versiones comerciales, las cuales son raramente accesibles para la pequeña y mediana empresa de comercio electrónico.

En este contexto, el proyecto europeo SME E-Compass [1] nace con el objetivo de generar herramientas de analítica web avanzadas y accesibles para las PYMES europeas. Estas aplicaciones software se nutren de diferentes fuentes de datos de traza web provenientes de huella electrónica (en forma de código *JavaSript*) integrada en las propias tiendas online. Sin embargo, la integración de información proveniente de múltiples fuentes de datos supone el tratamiento de modelos de datos diferentes, con esquemas y lenguajes de consulta heterogéneos.

Con esta motivación, proponemos en este trabajo un modelo semántico guiado por ontología para la recolección y el fusionado de datos de manera coherente, provenientes de traza web de servicios comerciales de huella electrónica. Como consecuencia, los datos procesados son anotados semánticamente y almacenados para su posterior utilización en el entrenamiento de algoritmos de minería de datos que analizan el comportamiento de los visitantes en sitios reales de comercio electrónico.

Nuestro modelo semántico utiliza una ontología de dominio como esquema de mediación, para la representación y consolidación de la semántica particular de los datos de traza web. Para la correspondencia entre los esquemas particulares de cada fuente de datos y nuestra ontología, hemos desarrollado una serie de funciones de *"mapeo (mappings)"* mediante las que se transforman los datos originales al formato estándar RDF (Resource Description Framework) [2]. De esta forma, todos los flujos de datos heterogéneos se almacenan en un repositorio RDF, sobre el cual se ofrende un servicio unificado de consulta para los algoritmos de análisis de alto nivel.

Por tanto, como aportación principal y original de este trabajo, hemos diseñado e implementado por primera vez una ontología OWL (Web Ontology Language) [1,4] para analítica web. Esta ontología contempla un conjunto de atributos y métricas de traza web, lo suficientemente exhaustivo y complementario, provenientes de las herramientas más representativas y utilizadas hoy en día para el análisis web: Google Analytics y Piwik.

Como aportación práctica, hemos validado nuestro modelo semántico mediante la captura e integración automática de diferentes flujos de datos de huella electrónica (Google Analytics y Piwik) alojada en 15 tiendas online reales de diferentes sectores comerciales (moda, belleza, turismo, electrónica, farmacia, gastronomía y venta al por menor en general) y países (Reino Unido, Grecia,

[1] SME-Ecompass FP7 European initiative http://www.sme-ecompass.eu/
[2] RDF in W3C https://www.w3.org/RDF/

Alemania y España), durante varios meses de actividad. Estos datos son por tanto integrados bajo un formato común y almacenados en un único repositorio RDF. Realizamos además varios casos de uso de aplicación, utilizando los datos integrados y anotados semánticamente, para el entrenamiento de algoritmos de minería de datos avanzados para el análisis del visitante web. En concreto, mediante estos algoritmos realizamos el análisis del comportamiento del visitante y su preferencia respecto a varios productos de una determinada marca comercial.

Este trabajo se organiza de la siguiente manera. En la siguiente Sección se presenta el modelo semántico propuesto, dando detalles de nuestra ontología, las fuentes de datos y el repositorio RDF desarrollado. En la Sección 3, se describe el caso de uso realizado para la validación del modelo semántico. Finalmente, la Sección 4 contiene las conclusiones y el trabajo futuro.

2. Modelo Semántico

El principal objetivo de este trabajo es recopilar, limpiar, consolidar e integrar información de diferentes fuentes de huella electrónica. Para ello se ha diseñado un modelo semántico cuyo elemento principal es una ontología que representa el conocimiento común del dominio de aplicación. En concreto, hemos utilizado la metodología "Ontology 101 development process"[5] para definir una ontología OWL que describe las principales características de las tiendas virtuales en siete pasos:

1. *Determinar el dominio y el ámbito de la ontología.* Inicialmente se tuvieron en cuenta las posibles variables de Google Analytics y de Piwik, además de las de los competidores de la tienda virtual, que son necesarias para los algoritmos de minería de datos. Por ejemplo: el origen y los atributo de los visitantes, los detalles de los productos y de los clientes, etc.
2. *Considerar la reutilización de ontologías ya existentes.* No hemos encontrado ontologías similares que se hayan desarrollado previamente para el análisis de datos de huella digital en comercio electrónico. Sin embargo, hemos tenido en cuenta parcialmente dos ontologías relacionadas: *GoodRelations* [3], que define un vocabulario estándar para comercio electrónico y la *Product Ontology*, que categoriza el tipo de producto basándose en Wikipedia.
3. *Enumerar los términos importantes en la ontología.* Los términos más importantes en la ontología se extrajeron en una fase previa de especificación de requisitos [2] a partir del conjunto mínimo de variables que se necesitan. Ejemplos de estos términos son: *Address*, *Visitor*, *Customer*, *Device*, *Browser*, *Geographical_origin*, *Number_of_visitors*, *Conversion_rate*, etc.
4. *Definir las clases y la jerarquía de clases.* A partir de la lista de términos importantes, obtenemos las clases de la ontología. La Figura 1 muestra el primer nivel de la jerarquía de clases, partiendo de la clase *Thing* ($\top$). Estas clases se relacionan con otras y algunas de ellas tienen subclases. Por ejemplo, *Bounce_rate*, *Total_revenue*, *Number_of_returning_visitors*, y *Number_of_transactions* son subclases de la clase *Analytic_parameters*.

5. *Definir las relaciones entre clases y sus atributos.* Para identificar las relaciones entre las clases (*object properties*) y los atributos de las clases (*data properties*) hemos tenido en cuenta el conjunto inicial de variables identificadas en el paso 1. Ejemplos de relaciones entre clases son: *an e-shop owner is owner of an e-shop*, *a visitor makes visits*, *a device has a browser*, *an IP address belongs to an organization*, etc. Ejemplos de atributos de clase son: *title and URL de una page*, *first and last name de un e-shop owner*, *version del operating system*, *duration de una visit*, etc. Para las subclases de *Analytic_parameters* se ha definido una relación para establecer la clase dominio de la misma. Por ejemplo, *Page* se relaciona con *Bounce_rate* y *Date_of_last_visit*; *E-shop* se relaciona con *Number_of_customers*. Los cuadros 2, 3, 5, and 6, describen un conjunto de las relaciones y atributos de las principales clases de la ontología.
6. *Definir las propiedades de los atributos.* Definición de las restricciones de cardinalidad y de valor (*value restrictions*). En nuestra ontología, las restricciones de valor se usan para especificar el tipo de datos válido en cada una de las *data properties* definidas para las subclases de *Analytic_parameters*. Por ejemplo, el rango de la propiedad *hasValue* se restringe a *float*, cuando su dominio es la clase *Bounce_rate*; el rango de la propiedad *hasValue* se restringe a *date*, cuando su dominio es la clase *Date_of_last_visit*.
7. *Crear las instancias (individuos).* Las instancias (individuos en OWL) corresponden a los datos de huella digital que se obtienen de Google Analytics, de Piwik, o del módulo de *srapping* de los competidores, para cada una de las tiendas virtuales. Estos datos se mapean a RDF teniendo en cuenta la ontología. También se han definido individuos para determinar los elementos específicos con los que se pueden relacionar algunas clases. Por ejemplo, cuando el dominio de la propiedad *hasType* es la clase *Article_number* su rango se restringe a los valores: "ASIN", "EAN", o "ISBN". "ASIN", "EAN", e "ISBN"se definen por tanto como instancias de la ontología.

2.1. Ontología

Como resultado del desarrollo anterior, nuestra ontología contiene un total de 62 clases (grupos de individuos que intercambian los mismos atributos), 61 propiedades de objeto (relaciones binarias entre los individuos), 33 restricciones de axioma y 3 individuos. La ontología completa, a la cual llamamos "wao.owl" (Web Analytics Ontology), puede consultarse a través de su enlace público en WebProtégé [3].

Por simplicidad, nos restringimos en este trabajo a la descripción de un subconjunto de las clases principales, incluyendo algunas de sus propiedades de objeto y de datos más interesantes y representativas. Estas clases son: *Analytics_parameters*, *E-shop*, *Visitor*, *Page* e *Item*. Cada una de estas clases requiere un conjunto de propiedades o condiciones para su contextualización, es decir, los individuos que satisfacen estas propiedades son miembros de estas clases.

[3] URL `http://stanford.io/1XHhHzr`

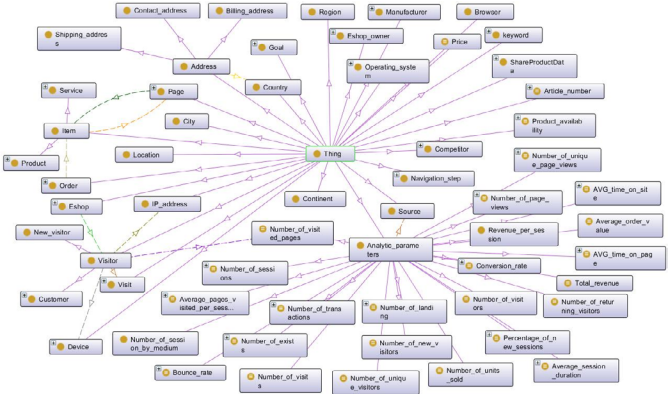


Figura 1. Vista general de la ontología WAO. Las flechas continuas se refieren a *subclase de(owl:subclassOf.* Las flechas discontinuas indican propiedades específicas.

- *Analytics_parameters*. Atributos dependientes del tiempo que proveen Google Analytics y Piwik. Cara uno de estos parámetros tiene un valor (*hasValue* en Cuadro 1) que corresponde con el dato que provee la herramienta de analítica web. Además tiene una fecha (*hasDate*), que corresponde con el día y la hora en que se genera el dato. Las principales subclases de los parámetros de análisis ($\sqsubseteq$ Analytic_parameters) son, entre otras: *Average_order_value*, *Average_pages_visited_per_session*, *Average_session_duration*, *Average_time_on_site*, *Bounce_rate*, *Conversion_rate*, *Number_of_transactions*, *Number_of_landings*, *Number_of_new_visitors*, *Number_of_page_views*, *Revenue_per_ session* y *Total_revenue*. El Cuadro 1 contiene algunas propiedades de objeto y de datos más representativas de *Analytics_parameters*, perteneciendo cada parámetro analítico a un tipo de datos. Por ejemplo, el tipo referente al número de transacciones en la tienda online (*Number_of_transactions*) es un número entero no negativo y el valor del ratio de conversión (*Conversion_rate*) es un número real. Las restricciones de tipo de datos se incluyen mediante las propiedades de datos.

- *E-shop.* Una *E-shop* o tienda online tiene varias páginas (Pages) y un propietario (e-shop's owner). La e-shop tiene atributos como la latitud, la longitud y la zona horaria. El propietario puede tener competidores, que son a su vez propietarios de otras e-shops. Los parámetros de análisis de una e-shop son: *average_order_value*, *average_pages_visited_per_ session*, *average_ session_duration*, *average_time_on_site*, *conversion_rate*, *date_of_last_transaction*, *number_of_customers*, *number_of_failed_transactions*, *number_of_successful_transactions*, *number_of_ new_customers*, *number_of_new_visitors, number_of_sessions_by_medium, num-*

Cuadro 1. Grupo Analytics_parameters: las propiedades de objeto y de datos se representan en lógica de descripciones

Propiedades de Objeto	Lógica de Descripciones
hasBrowser	∃ hasBrowser.Thing ⊑ Analytic_parameters ⊔ Device ⊤ ⊑ ∀ hasBrowser.Browser
hasCity	∃ hasCity.Thing⊑ Analytic_parameters ⊔ Location ⊔ Visitor ⊤ ⊑ ∀ hasCity.City
hasRegion	∃ hasRegion.Thing ⊑ Analytic_parameters ⊔ Location ⊔ Visitor ⊤ ⊑ ∀ hasRegion.Region
hasCountry	∃ hasCountry.Thing ⊑ Analytic_parameters ⊔ Location ⊔ Visitor ⊤ ⊑ ∀ hasCountry.Country
hasContinent	∃ hasContinent.Thing ⊑ Analytic_parameters ⊔ Location ⊤ ⊑ ∀ hasContinent.Continent
hasSource	∃ hasSource.Thing ⊑ Analytic_parameters ⊤ ⊑ ∀ hasSource.Source
Propiedades de Datos	**Lógica de Descripciones**
hasDate	∃ hasDate.DatatypeLiteral ⊑ Analytic_parameters ⊔ Price ⊔ Product_availability ⊤ ⊑ ∀ hasDate.DatatypedateTimeStamp
hasHour	∃ hasHour.DatatypeLiteral ⊑ Analytic_parameters ⊤ ⊑ ∀ hasHour.Datatypetime
hasNetworkDomain	∃ hasNetworkDomain.DatatypeLiteral ⊑ Analytic_parameters ⊤ ⊑ ∀ hasNetworkDomain.Datatypestring
hasValue	∃ hasValue.DatatypeLiteral ⊑ Analytic_parameters ⊔ Article_number ⊔ Price ⊔ Product_availability

ber_of_transactions, number_of_unique_visitors, number_of_units_sold, number_of_visitors, number_of_visits, percentage_of_new_sessions, revenue_per_session, total_revenue y *number_of_returning_visitors*. Todos estos parámetros relacionados con la tienda online son dependientes de la dimensión temporal. Por tanto, estos parámetros se modelan como clases que se relacionan con la e-shop mediante las correspondientes propiedades de objeto. En el Cuadro 2 se recoge un subconjunto de propiedades de objeto y de datos con clases del grupo E-shop como dominio.

- *Visitor/Visit*. La clase visitante (*Visitor*) tiene dos subclases: *Customer*, referente al cliente y *New_visitor*, que anota al nuevo visitante en la tienda online. Un cliente es un visitante que compra. Los clientes tienen nombre y dirección postal (provenientes del registro en la web), mientras que los visitantes no. Un visitante entra en la tienda online a través de un dispositivo (*Device*). Los parámetros de análisis para los visitantes son: *bounced_rate, number_of_visits* y *number_of_visited_pages*, mientras que el cliente tiene además el parámetro *number_of_transactions*.

Los visitantes acceden a páginas (*Pages*), por lo que efectúan visitas. Estas visitas son fundamentales para capturar el comportamiento del visitante. Cada visita tiene una página de entrada y otra de salida (que puede ser la misma). Además, también tiene una página de referencia o de enlace, a partir de la cual el visitante entra en la web: buscadores, redes sociales, webs con enlaces de publicidad, etc. Si esta página de referencia es un buscador, se pueden obtener las palabras clave utilizadas para realizar la búsqueda. Las visitas también tienen una duración, cuyos atributos dependen de la marca temporal desde que se accede a la página de entrada hasta que se sale por la página de salida; un indicador de si se ha realizado compra y el número total de artículos vendidos;

Cuadro 2. Grupo Eshop: las propiedades de objeto y de dato se representan en lógica de descripciones

Propiedades de Objeto	Lógica de Descripciones
hasVisitor	$> \equiv$ makesVisit$>^{-}$
	$\exists$ hasVisitor Thing $\sqsubseteq$ Eshop
	$\top \sqsubseteq \forall$ hasVisitor Visitor
hasNumberOfVisitors	$\exists$ hasNumberOfVisitors Thing $\sqsubseteq$ Eshop $\sqcup$ Page
	$\top \sqsubseteq \forall$ hasNumberOfVisitors Number_of_visitors
hasNumberOfVisits	$\exists$ hasNumberOfVisits Thing $\sqsubseteq$ Eshop $\sqcup$ Page $\sqcup$ Visitor
	$\top \sqsubseteq \forall$ hasNumberOfVisits Number_of_visits
isOwnerOf	$\exists$ isOwnerOf Thing $\sqsubseteq$ Eshop_owner
	$\top \sqsubseteq \forall$ isOwnerOf Eshop
Propiedades de Datos	Lógica de Descripciones
hasName	$\exists$ hasName DatatypeLiteral $\sqsubseteq$ Browser $\sqcup$ Competitor $\sqcup$ Eshop $\sqcup$ Goal $\sqcup$ Item $\sqcup$ Operating_system $\sqcup$ Page $\sqcup$ Product
	$\top \sqsubseteq \forall$ hasName Datatypestring
hasURL	$\exists$ hasURL DatatypeLiteral $\sqsubseteq$ Competitor $\sqcup$ Eshop $\sqcup$ Page $\sqcup$ Price
	$\top \sqsubseteq \forall$ hasURL Datatypestring

Cuadro 3. Grupo Visitor: las propiedades de objeto y de datos se representan en lógica de descripciones

Propiedades de Objeto	Lógica de Descripciones
hasDevice	$\exists$ hasDevice Thing $\sqsubseteq$ Visitor
	$\top \sqsubseteq \forall$ hasDevice Device
hasNumberOfVisits	$\exists$ hasNumberOfVisits Thing $\sqsubseteq$ Eshop $\sqcup$ Page $\sqcup$ Visitor
	$\top \sqsubseteq \forall$ hasNumberOfVisits Number_of_visits
hasCity	$\exists$ hasCity.Thing$\sqsubseteq$ Analytic_parameters $\sqcup$ Location $\sqcup$ Visitor
	$\top \sqsubseteq \forall$ hasCity.City
makesVisit	hasVisitor¿ $\equiv$ makesVisit¿$^{-}$
	$\exists$ makesVisit Thing $\sqsubseteq$ Visitor
	$\top \sqsubseteq \forall$ makesVisit Visit
Propiedades de Datos	Lógica de Descripciones
hasDaysSinceFirstVisit	$\exists$ hasDaysSinceFirstVisit DatatypeLiteral $\sqsubseteq$ Visitor
	$\top \sqsubseteq \forall$ hasDaysSinceFirstVisit DatatypenegativeInteger
hasDaysSinceLastOrder	$\exists$ hasDaysSinceLastOrder DatatypeLiteral $\sqsubseteq$ Visitor
	$\top \sqsubseteq \forall$ hasDaysSinceLastOrder DatatypenegativeInteger
hasDaysSinceLastVisit	$\exists$ hasDaysSinceLastVisit DatatypeLiteral $\sqsubseteq$ Visitor
	$\top \sqsubseteq \forall$ hasDaysSinceLastVisit DatatypenegativeInteger
	$\exists$ IsReturningVisitor DatatypeLiteral $\sqsubseteq$ Visitor
	$\top \sqsubseteq \forall$ IsReturningVisitor Datatypeboolean

un número de acciones, de eventos y de búsquedas. Durante una visita se realizan transacciones. Un atributo importante de la visita es la ruta, que comprende las páginas anteriores y siguiente en el acceso a la web. De hecho, la clase "pasos de navegación" *Navigation_step* se utiliza para modelar las rutas que siguen los visitantes a lo largo de sus visitas. Los Cuadros 3 y 4 contienen las propiedades que tienen dominios las clases visitante y visita, respectivamente.

- *Page.* Una página en una web de comercio electrónico contiene *items*, es decir, productos y/o servicios a la venta. Los parámatros de análisis de la clase *Page* son: *average_order_value, average_time_on_page, bounce_rate, date_of_last_ visit, number_of_exits, number_of_landings, number_of_new_visitors, number_of_page_views, number_of_returning_visitors, number_of_sessions_by_medium* (los medios pueden ser: enlaces directos, redes sociales y motores de búsqueda), *number_of_sessions, number_of_unique_page_views, number_of_unique_visitors, number_of_units_sold, number_of_visitors, number_of_visits, revenue_per_session and*

Cuadro 4. Grupo Visit: las propiedades de objeto y de datos se representan en lógica de descripciones

Propiedades de Objeto	Lógica de Descripciones
hasNavigationStep	$\exists$ hasNavigationStep.Thing $\sqsubseteq$ Visit
	$\top \sqsubseteq \forall$ hasNavigationStep.Navigation_step
hasRefererKeyword	$\exists$ hasRefererKeyword.Thing $\sqsubseteq$ Visit
	$\top \sqsubseteq \forall$ hasRefererKeyword.Referer_keyword
makesVisit	hasVisitor¿ $\equiv$ makesVisit¿$^-$
	$\exists$ makesVisit.Thing $\sqsubseteq$ Visitor
	$\top \sqsubseteq \forall$ makesVisit.Visit
Propiedades de Datos	Lógica de Descripciones
hasDuration	$\exists$ hasDuration.DatatypeLiteral $\sqsubseteq$ Visit
	$\top \sqsubseteq \forall$ hasDuration.Datatypetime
hasReturningVisitor	$\exists$ hasReturningVisitor.DatatypeLiteral $\sqsubseteq$ Visit
	$\top \sqsubseteq \forall$ hasReturningVisitor.Datatypeboolean

Cuadro 5. Grupo Page: las propiedades de objeto y de datos se representan en lógica de descripciones

Propiedades de Objeto	Lógica de Descripciones
hasNumberOfVisits	$\exists$ hasNumberOfVisits Thing $\sqsubseteq$ Eshop $\sqcup$ Page $\sqcup$ Visitor
	$\top \sqsubseteq \forall$ hasNumberOfVisits Number_of_visits
hasNumberOfVisitors	$\exists$ hasNumberOfVisitors Thing $\sqsubseteq$ Eshop $\sqcup$ Page
	$\top \sqsubseteq \forall$ hasNumberOfVisitors Number_of_visitors
hasTotalRevenue	$\exists$ hasTotalRevenue Thing $\sqsubseteq$ Eshop $\sqcup$ Page
	$\top \sqsubseteq \forall$ hasTotalRevenue Total_revenue
isOnPage	$\exists$ isOnPage Thing $\sqsubseteq$ Item
	$\top \sqsubseteq \forall$ isOnPage Page
Propiedades de Datos	Lógica de Descripciones
hasName	$\exists$ hasName DatatypeLiteral $\sqsubseteq$ Browser $\sqcup$ Competitor $\sqcup$ Eshop $\sqcup$ Goal $\sqcup$ Item $\sqcup$ Operating_system $\sqcup$ Page $\sqcup$ Product
	$\top \sqsubseteq \forall$ hasName Datatypestring
hasURL	$\exists$ hasURL DatatypeLiteral $\sqsubseteq$ Competitor $\sqcup$ Eshop $\sqcup$ Page $\sqcup$ Price
	$\top \sqsubseteq \forall$ hasURL Datatypestring
hasTitle	$\exists$ hasTitle DatatypeLiteral $\sqsubseteq$ Page
	$\top \sqsubseteq \forall$ hasTitle Datatypestring

total_revenue. Los atributos de una página son básicamente el título y la URL. Cuadro 5 contiene una serie de propiedades representativas del dominio de página. Una propiedad interesante es *hasTotalRevenue*, que en esta tabla hace referencia a los ingresos generados en la propia página, aunque puede utilizarse también para toda la e-shop, ya que se puede calcular para ambos casos.

- Item. Un Item es un producto o servicio ofrecido en una tienda online. Los items específicos de una e-shop se modelan mediante una ontología de dominio específica, es decir, viajes, libros, música, etc. La tabla del Cuadro 6 describe algunas propiedades de objeto y de datos más representativas de esta clase. De acuerdo con estas propiedades, un Item tiene un precio (*hasPrice*), que es válido durante cierta fecha. Por tanto, los atributos para precio son: valor (*value*), moneda (*currency*) y fecha de validez. Los atributos para Item son su categoría y un indicador de si han sido o no eliminados de la tienda. Los productos tienen un fabricante (*manufacturer*) y entre sus atributos cuenta con: el nombre, el tipo y su disponibilidad en fecha y referencia específica. El número de artículo puede ser uno de los códigos estándares: “ASIN”, “EAN” o “ISBN”.

Cuadro 6. Grupo Item: las propiedades de objeto y de dato se representan en lógica de descripciones

Propiedades de Objeto	Lógica de Descripciones
hasItem	$\exists$ hasItem Thing $\sqsubseteq$ Page
	$\top \sqsubseteq \forall$ hasItem Item
hasPrice	$\exists$ hasPrice Thing $\sqsubseteq$ Item $\sqcup$ ShareProductData
	$\top \sqsubseteq \forall$ hasPrice Price
includes	$\exists$ includes Thing $\sqsubseteq$ Order
	$\top \sqsubseteq \forall$ includes Item
isOnPage	$\exists$ isOnPage Thing $\sqsubseteq$ Item
	$\top \sqsubseteq \forall$ isOnPage Page
Propiedades de Datos	Lógica de Descripciones
hasCategory	$\exists$ hasCategory DatatypeLiteral $\sqsubseteq$ Item
	$\top \sqsubseteq \forall$ hasCategory Datatypestring
hasName	$\exists$ hasName DatatypeLiteral $\sqsubseteq$ Browser $\sqcup$ Competitor $\sqcup$ Eshop $\sqcup$ Goal $\sqcup$ Item $\sqcup$ Operating_system $\sqcup$ Page $\sqcup$ Product
	$\top \sqsubseteq \forall$ hasName Datatypestring
hasItemID	$\exists$ hasItemID DatatypeLiteral $\sqsubseteq$ Item
	$\top \sqsubseteq \forall$ hasItemID DatatypenonNegativeInteger
hasQuantity	$\exists$ hasQuantity DatatypeLiteral $\sqsubseteq$ Item
	$\top \sqsubseteq \forall$ hasQuantity DatatypenonNegativeInteger

2.2. Fuentes de Datos

Como hemos mencionado anteriormente, hemos seleccionado dos tipos principales fuentes de datos provenientes de servicios de traza web o huella electrónica ofrecidos por las aplicaciones Google Analitics y Piwik. Estas herramientas fueron seleccionadas tras una serie de entrevistas y encuestas a un conjunto de más de 150 propietarios de tiendas online de diferentes países y sectores comerciales. Como resultado se generó un informe [2] donde se obtiene, entre otras conclusiones, que la mayoría de las empresas entrevistadas (68 %) utilizan Google Analytics para hacer sus análisis de visitas, seguido por Piwik (16 %) y el resto de herramientas en el mercado. No obstante, si bien Google Analitics obtiene un gran número de atributos de visita, estos dados se sirven de forma muy agregada y los valores references a comercio sólo están disponibles en su versión de pago. Por tanto, como complemento a esta fuente de información, también utilizamos la plataforma Piwik, ya que nos ofrece una gran cantidad de atributos desagregados, además de información de comercio (ventas, ganancias, etc.) de manera abierta (sin necesidad de pago).

Como fuente adicional, también se contemplan los datos que se generan mediante procesos dedicados de barrido web, conocidos como *Web Scrapping*, que obtienen información de una selección de webs de competidores para el seguimiento de precios y novedades de productos.

El proceso de recolectar los datos de diferentes fuentes y transformarlos en RDF se lleva a cabo mediante las funciones de mapeo. Para cada fuente se ofrece un conjunto diferente de métodos para obtener, armonizar y almacenar los datos en RDF de acuerdo a la ontología diseñada. De hecho, se han desarrollado un gran número de funciones de mapeo correspondientes a los atributos de análisis web, en referencia a cada clase de la ontología. Sin embargo, aquellos atributos que comparten una estructura común se han mapeado utilizando funciones genéricas, por tanto aprovechando el diseño eficiente de la ontología.

- ***Google Analytics***[4]. El proceso traza web, o web tracking, en Google Analytics se realiza mediante un "Snippet" o código de huella electrónica que proporciona una API de funciones de acceso para cada valor de atributo y métrica. Esta "huella electrónica" consiste realmente en un pequeño código de JavaScript que se aloja en el fuente HTML de la tienda online y se despliega en el servicio web donde se hospeda la e-shop. Esta pieza de código activa el tracking de Google Analytics mediante el proceso de JavaScript `ga.js`/`analytics.js`. Las funciones de mapeo instancian este componente para la obtención de los datos, para formatearlos inmediatamente en RDF de manera sistemática. Los atributos utilizados, así como sus combinaciones, representan un conjunto lo suficientemente representativo y amplio para cubrir los requisitos de análisis. Cabe destacar que la ontología se puede extender para considerar cualquier atributo.

- ***Piwik*** [5] es una aplicación de analítica web de código abierto que se despliega en un servicio web PHP/MySQL. El proceso de tracking web se realiza en Piwik de manera similar a Google Analytics, es decir, mediante la utilización de un trozo de código de huella electrónica alojado en la tienda virtual. En el caso de Piwik, hemos desplegado nuestro propio servicio de administración, por lo que los datos de traza web se almacenan en una base de datos relacional (MySQL). Por tanto, las funciones de mapeo consultan directamente esta base de datos y transforman los valores de los atributos en formato RDF, siguiendo la estructura lógica marcada por nuestra ontología.

- ***Web Scrappers***. Entre las aplicaciones desarrolladas en el proyecto SME E-Compass, contamos además con una serie de métodos para automatizar el "rastreo" web, mediante los cuales obtenemos información en tiempo real sobre los precios y los productos en determinadas tiendas online de competidores. Esta funcionalidad ofrece un servicio de datos sobre los competidores en formato JSON[6]. En este caso, nuestros *mappings* convierten la información desde JSON a RDF siguiendo el modelo semántico especificado por la ontología.

2.3. Repositorio RDF

Toda la información procesada se consolida en un repositorio RDF que integra las distintas fuentes de datos de analítica web. Este repositorio está conectado además con un servicio de SPARQL Endpoint, a través del cual podemos hacer consultas sobre los datos de huella electrónica y scrapping de manera deambiguada e independientemente de la fuente de origen.

Como ejemplo de acceso a los datos, consideremos un escenario en el que un algoritmo de análisis precisa de información referente a las visitas de una tienda online determinada, en cierta fecha y/o periodo de tiempo. La información requerida consistiría tanto en datos desagregados de visitas, como aquellos proporcionados por Piwik, así como en métricas calculadas, como las proporcionadas por Google Analytics.

[4] http://www.google.com/analytics/
[5] http://piwik.org/
[6] http://json.org

Consulta SPARQL Q1:

```
PREFIX
vis:<http://www.sme-ecompass.eu/ontologies/visitor_behaviour.owl#>
SELECT ?e as ?eshop, ?fat as ?date, ?vi as ?visit,
?vts as ?visit_total_searches, ?vte as ?visit_total_events,
?vd as ?visit_duration, ?vgc as ?visit_total_goal_converted,
?bv as ?total_bounce_rate, ?cv as ?total_conversion_rate,
?lv as ?total_number_of_entries, ?nv as ?total_number_of_new_visitors
FROM
<http://www.sme-ecompass.eu/ontologies/visitor_behaviour/<eshop_id>/>
WHERE{
   ?e vis:hasVisitor ?vt.
   ?vt vis:makesVisit ?vi.
   ?vi vis:hasFirstActionTime ?fat.
   ?vi vis:hasNumberOfSearches ?vts.
   ?vi vis:hasNumberOfEvents ?vte.
   ?vi vis:hasDuration ?vd.
   ?vi vis:hasTotalGoalConverted ?vgc.
   ?e vis:hasBounceRate ?b.
   ?b vis:hasValue ?bv.
   ?b vis:hasDate ?d.
   ?e vis:hasConversionRate ?c.
   ?c vis:hasValue ?cv.
   ?c vis:hasDate ?d.
   ?e vis:hasNumberOfLandings ?l.
   ?l vis:hasValue ?lv.
   ?l vis:hasDate ?d.
   ?e vis:hasNumberOfNewVisitors ?n.
   ?n vis:hasValue ?nv.
   ?n vis:hasDate ?d.
   FILTER(str(?fat) > "2015-10-23" && str(?fat)
   < "2015-10-24" && str(?d) = "2015-10-23")
}
```

La consulta SPARQL Q1 unifica la lógica de acceso a partir de la cual se obtienen los resultados de ejemplo resumidos en el Cuadro 7. Estos resultados se corresponden con dos visitas consecutivas a la tienda con ID `<eshop-id>`, que se realizaron con fecha 2015-10-23. Los IDs de las visitas son 75688 y 75692, las cuales se capturaron con marca temporal 14:19:44 y 14:21:41, respectivamente. Tal y como se muestra en la tabla, la visita con tiempo más prolongado desembocó en un objetivo de conversión (una venta efectiva), mientras que la visita de corta duración terminó sin ningún tipo de conversión, lo cual representa al visitante que abandona la tienda de manera prematura.

En el caso de los atributos agregados, éstos son calculados para todas las visitas en el periodo de tiempo establecido en la consulta SPARQL. Por lo tanto, como se muestra en la segunda mitad del Cuadro 7, la tienda registra una "tasa

Cuadro 7. Dos ejemplos del resultado de la consulta SPARQL Q1 para un determinado intervalo temporal (día 2015-10-23) de una tienda online real

Atributo/Métrica	Visit75688	Visit75692
timestamp	14:19:44	14:21:41
visit_total_searches	0	0
visit_total_events	0	0
visit_total_duration	2071	12
visit_total_goal_converted	1	0
total_bounce_rate	52.6066	
total_conversion_rate	34.1232	
total_number_of_entries	211	
total_number_of_new_visitors	145	

de rebote" (*bounce rate*, se refiere a la proporción de visitantes que entran en un sitio web y lo abandonan después de haber visto una sola página web, en unos pocos segundos) cercana al 53 %, con un ratio de conversión del 34,12 %. Estos porcentajes corresponden a todas las visitas, rebotes y compras en la tienda durante el periodo temporal especificado en la consulta. Otro atributo de especial interés es el número de nuevos visitantes, que para la tienda examinada y el periodo de tiempo consultado, es de 145, es decir, el 68.72 % sobre el conjunto total de visitas en la web. Toda esta información puede ser ahora utilizada en conjunto para entrenar algoritmos predictivos de minería de datos, con el objetivo de obtener indicios que ayuden al comerciante a adoptar una determinada estrategia de marketing para captar nuevos clientes.

En este sentido, el repositorio RDF propuesto incorpora además un servicio de funciones REST API que implementan consultas SPARQL predefinidas. De esta forma se automatiza y se simplifica el acceso a la información almacenada. Los datos proporcionados por estas funciones de consulta son entonces utilizados como entrada para los algoritmos minería de datos que realizan los procesos de análisis. En la siguiente sección, pasamos a describir un caso de uso típico de análisis del visitante en términos de validación de nuestro modelo semántico.

3. Validación: Caso de Uso

El análisis de productos permite comprender mejor la relación entre visitantes y artículos comprados, que en función de los resultados, se traducirán en actuaciones sobre el precio o el posicionamiento del producto para mejorar su visibilidad en la web. El análisis en este sentido permite conocer la relación entre los visitantes que buscan cierto producto en la tienda y qué proporción de ellos terminan comprando. Nos centramos en este caso de uso en el análisis de una tienda online real de Alemania dedicada a la venta de artículos de belleza.

La información obtenida puede ayudar a optimizar las ventas. Si un producto es raramente visitado, pero la tasa de conversión es alta, podríamos colocarlo en un lugar más prominente en el sitio web para mejorar las ventas y actuar como "gancho" para otros artículos. Un producto que tiene pocos visitantes y bajo ratio de conversión, debe ser inspeccionado en cuanto al precio o incluso sustituirlo o eliminarlo por completo.

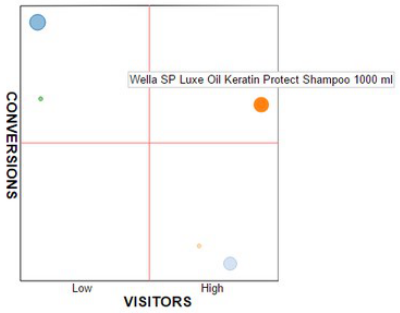


Figura 2. Análisis mediante gráfica DAFO de visitantes y productos

Este tipo de análisis requiere de información, tanto de actividad respecto a la navegación del visitante, como sobre los hábitos de compra. Por tanto, para este caso de uso nos centramos en datos capturados respecto a atributos de Piwik, como: *idaction_sku* (Stock-keeping unit), *idaction_name, idaction_category, location_geoip, visit_first_action_time, visitor_days_since_order, visit_goal_buyer* y *visit_goal_converted*. Estos atributos se modelan mediante nuestra ontología según las descripciones de la Sección 2.1 para la clase *Visit*.

El modelo de minería utiliza en primer lugar, un algoritmo de aprendizaje no-supervisado: clustering para generar grupos de visitantes y productos por comportamiento; y en segundo lugar una técnica de aprendizaje supervisado mediante árboles de decisión, para asignar los nuevos visitantes y productos a sus correspondientes grupos.

Como resultado, La Figura 2 muestra las relaciones entre conversiones de productos y visitantes. Se trata de un gráfico DAFO (Debilidades, Amenazas, Fortalezas y Oportunidades) en el cual, el eje horizontal indica la cantidad de visitantes para cierto producto, mientras que el eje vertical muestra el ratio de conversión de dicho producto. El tamaño de los puntos representan los ingresos totales producidos por el producto. Por tanto, se pueden establecer diferentes estrategias comerciales por las indicaciones de cada cuadrante. Los productos con pocos visitas y bajo ratio de conversión denotan poco interés por parte de los visitantes, con pocas ventas. Los productos más rentables se localizan en el cuadrante con pocas visitas pero alto número de conversiones, por lo que registran poco tráfico web. En este sentido, con cierta actividad promocional adicional, este producto podría obtener incluso mayores ventas. Aquellos productos con muchas visitas pero pocas ventas son candidatos típicos para revisar su posición en la web, su precio, etc.

Por último, los productos con muchas visitas y alto ratio de conversión denotan artículos atractivos y rentables (véase el ejemplo de "Wella SP Luxe Aceite Queratina Protect Champoo 1000 ml" en la figura). La estrategia global sería entonces intentar posicionar nuestros productos en el cuadrante superior de la derecha referente a productos con gran número de visitas y ventas.

4. Conclusiones

En este trabajo, proponemos un repositorio RDF para recopilar, integrar y almacenar información de traza web procedente de distintas fuentes de huella electrónica. Estas se consolidan en el repositorio diseñado para proporcionar semántica común a los datos y dar servicio homogéneo a algoritmos de Minería de Datos. El servicio propuesto se ha validado mediante traza digital real (Google Analitics y Piwik) de 15 tiendas virtuales de diferentes sectores y países europeos (UK, España, Grecia y Alemania) durante varios meses de actividad. En concreto, se presenta un caso de uso real sobre el análisis de visitas y conversiones en una tienda de productos de belleza.

Como aportación principal y original de este trabajo, hemos diseñado e implementado por primera vez una ontología OWL (Web Ontology Language) para analítica web. Esta ontología contempla un conjunto de atributos y métricas de traza web, lo suficientemente exhaustivo y complementario, provenientes de las herramientas más representativas y utilizadas hoy en día para el análisis web: Google Analytics y Piwik, además de la aplicación propia de Web Scrapping.

Como trabajo futuro, el siguiente paso consistirá en ampliar la ontología para considerar nuevas fuentes de analítica web de otros analizadores comerciales (Adobe Syte Catalyst, Yahoo WA, etc.). Además, estamos interesados en incorporar a nuestro repositorio otros conjuntos de Open Linked Data para enriquecer el modelo semántico con información que proporcione nuevas perspectivas al análisis: información meteorológica, tendencias de consumo, descripciones de productos, afinidades por sector social o geográfico, etc.

Referencias

1. M. Dean and G. Schreiber. OWL web ontology language reference. Technical report, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-ref-20040210/, Latest version available at http://www.w3.org/TR/owl-ref/, 2004.
2. SME E-Compass. D2.1 sme-e-compass requirements analysis. Technical report, Public Deliverable, 2004.
3. M. Hepp. Goodrelations: An ontology for describing products and services offers on the web. In *Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW2008)*, pages 332–347. Springer LNCS, Vol 5268, 2008.
4. D. McGuinness and F. Harmelen. OWL web ontology language overview. Technical report, W3C Recommendation, 2004.
5. Natalya F. Noy and Deborah L. McGuinness. Dontology development 101: A guide to creating your first ontology. Technical report, tanford University Knowledge Systems Laboratory Technical Report KSL-01-05, 2001.

v-RDFCSA: Compresión e Indexación de Colecciones de Versiones RDF

Ana Cerdeira-Pena[1], Antonio Fariña[1], Javier D. Fernández[2], and Miguel A. Martínez-Prieto[3]

[1] Laboratorio de Bases de Datos, Universidade da Coruña
acerdeira@udc.es, fari@udc.es
[2] Vienna University of Economics and Business (WU)
jfernand@wu.ac.at
[3] DataWeb Research, Universidad de Valladolid
migumar2@infor.uva.es

Resumen La gestión de grandes colecciones RDF es un tema de gran interés en la Web de Datos, pero cuyas técnicas más relevantes no van más allá de una simple visión estática de la colección, dejando al margen la problemática relacionada con su evolución temporal. Sin embargo, las colecciones evolucionan para actualizar la descripción de su dominio y, con ello, generan múltiples versiones que precisan un almacenamiento efectivo de cara a su explotación por diferentes tipos de aplicaciones semánticas. En este artículo proponemos una nueva técnica para la compresión de colecciones de versiones RDF. Nuestra propuesta (v-RDFCSA) extiende el autoíndice RDFCSA con estructuras de bits que codifican la información de versionado. De esta manera, preservamos los triples RDF en espacio comprimido y, sobre esta representación, resolvemos operaciones de consulta temporales basadas en patrones SPARQL. Nuestra evaluación muestra que v-RDFCSA reduce los requisitos de almacenamiento entre 35 y 60 veces respecto al estado del arte actual y consigue más de un orden magnitud de ventaja en la resolución de consultas.

1. Introducción

El uso de RDF (*Resource Description Framework*) [14] se ha generalizado durante la última década. Proyectos de datos abiertos, como *Linked Open Data*, o iniciativas colaborativas, como *schema.org*, han promovido el uso de RDF como estándar *de facto* para la descripción, en la Web, de entidades procedentes de diferentes campos de conocimiento. Colecciones RDF de datos biológicos, publicaciones científicas, multimedia o datos gubernamentales son sólo algunos ejemplos de la variedad que podemos encontrar en la *Web de Datos*.

La Web de Datos plantea un espacio de conocimiento que está creciendo de forma progresiva [12] y en el que, además, los contenidos publicados en cada colección evolucionan con el objetivo de describir los nuevos hechos que se producen a lo largo del tiempo. Por ejemplo, *DBpedia live*[4] actualiza diariamente sus contenidos para incorporar los cambios producidos durante ese día en Wikipedia. Estos cambios dan lugar a nuevas versiones de la colección que, en la

[4] http://live.dbpedia.org/

práctica, tienden a ser similares a sus predecesoras. La última versión es la que está vigente en un momento dado, pero también es necesario preservar las versiones previas con el objetivo de satisfacer necesidades como, por ejemplo, el análisis evolutivo los datos o la posibilidad de revertir los cambios realizados.

El presente problema es bastante reciente en la Web de Datos, pero tiene un precedente claro en la *World Wide Web*. En este caso, los archivos contienen múltiples versiones de páginas web y su gestión presenta problemas de escalabilidad importantes en su almacenamiento y explotación [9]. Aunque el nivel de escala que se presenta en la Web de Datos es, por el momento, inferior al de la WWW, el reto es comparable, y abre con ello una nueva línea de investigación relacionada con la preservación de colecciones históricas de RDF (referidas como *archivos RDF*) y la necesidad de poder consultar estos datos de una manera eficiente [6]. El estado del arte presenta algunas estrategias para afrontar este nuevo reto (ver Sección 2.1), pero ninguna de ellas puede considerarse efectiva atendiendo a las necesidades de almacenamiento que demandan. Por ejemplo, dichas estrategias utilizan hasta 15 veces más espacio que el que requiere el compresor *gzip* para almacenar un archivo RDF de referencia [8]. Estos números están muy alejados de los que se obtienen al utilizar técnicas específicas para la compresión de archivos web [4]. Dichas técnicas no sólo consiguen reducir drásticamente las necesidades de almacenamiento (en algunos casos, utilizan menos de un 2 % del espacio originalmente ocupado por el archivo), sino que también permiten realizar búsquedas eficientes sobre los datos comprimidos.

La brecha existente ente ambos escenarios fundamenta la investigación formulada en este trabajo, considerando que ya existen técnicas específicas para la compresión de RDF que, a su vez, facilitan la resolución eficiente de patrones básicos de consulta [7, 1, 2]. Sin embargo, todas estas técnicas se ciñen a la visión estática de la colección RDF y no han considerado todavía ningún tipo de solución para afrontar las necesidades subyacentes a la gestión de sus versiones.

En este artículo presentamos `v-RDFCSA`, la primera técnica específica para la compresión de archivos RDF que, a su vez, proporciona algoritmos eficientes para la resolución de operaciones de consulta sobre ellos. `v-RDFCSA` identifica el conjunto de todos los triples diferentes usados en el archivo y los comprime con RDFCSA [2], un *autoíndice* capaz de resolver consultas SPARQL sobre la representación comprimida de la colección. Por otro lado, `v-RDFCSA` utiliza secuencias de bits para codificar, de forma sucinta, la información de versionado. Nuestros experimentos, utilizando el *benchmark* BEAR [8], muestran que `v-RDFCSA` usa apenas 5,7 – 7,3GB de espacio para almacenar un archivo RDF de 325GB y resuelve las operaciones de consulta estudiadas un orden de magnitud más rápido que el *baseline* considerado.

El resto del artículo se organiza de la siguiente manera. En la Sección 2 planteamos una revisión general de los conceptos necesarios para entender nuestra propuesta, cuya explicación abordamos en la Sección 3. La Sección 4 muestra una evaluación experimental de `v-RDFCSA` y compara su rendimiento respecto al obtenido por un *baseline* de referencia. Finalmente, la sección 5 resume las conclusiones obtenidas con este trabajo y las líneas de trabajo futuro.

2. Trabajo Relacionado

Los archivos RDF utilizan dos tecnologías principales: el modelos de datos RDF y el lenguaje de consulta SPARQL. RDF [14] describe hechos en forma de estructuras ternarias, llamadas *triples* (o tripletas). Cada triple `(sujeto, predicado, objeto)` describe una propiedad (*predicado*) de un *sujeto*, otorgándole un valor (*objeto*) concreto. Por ejemplo, los triples `(JohnDoe, age, 45)` y `(JohnDoe, email, john@example.org)` establecen la edad y el correo electrónico del sujeto *JohnDoe*. En la práctica, cada uno de los triples puede verse como un grafo que conecta los nodos sujeto y objeto mediante una arista etiquetada con el predicado correspondiente. Por tanto, una colección de triples conforma un grafo etiquetado y dirigido que representa una base de conocimiento sobre un conjunto de entidades. Por su parte, SPARQL [11] es un lenguaje de consulta para grafos RDF, con una expresividad (y sintaxis) similar a SQL, pero basado en la correspondencia entre patrones de grafo. Una consulta SPARQL está formada por patrones de triples, esto es, triples en los que cada uno de los componentes puede ser una variable que se debe confrontar con el grafo RDF que se consulta. Operadores más complejos, como *joins*, uniones, patrones opcionales, filtros y otros modificadores de consulta, completan la funcionalidad de este lenguaje.

Archivo RDF. Un archivo RDF organiza las versiones de una colección RDF, anotando los triples con las versiones a las que pertenecen. Un *triple anotado con versión* [8] es, por tanto, un triple RDF `(sujeto, predicado, objeto)` con una etiqueta $i \in [1, \mathcal{N}]$ que representa la versión en la que es válido dicho triple. Por lo tanto, un *archivo RDF*, $\mathcal{A}$, es un conjunto de triples anotados con versión[5].

La Figura 1 ilustra un archivo RDF con 3 versiones en las que se describe información sobre jugadores y entrenadores del club de fútbol *"Atlético de Madrid"*. En la primera versión, V_1, se representan dos jugadores, `ex:Torres` y `ex:Simeone`, y que el entrenador del equipo es `ex:Manzano`. Ambos jugadores abandonan el equipo en la versión V_2, al tiempo que se contrata al jugador `ex:Falcao`. Para finalizar, en la última versión, V_3, `ex:Simeone` vuelve al equipo, pero en calidad de entrenador, reemplazando a `ex:Manzano`, mientras que `ex:Falcao` cede su puesto de jugador a `ex:Torres`, quien también regresa al equipo.

Funcionalidad de Consulta. Los archivos RDF proporcionan funcionalidad de consulta SPARQL sobre una o más versiones, o sobre las diferencias (*deltas*) existentes entre dos o más versiones dadas. Dichas funcionalidades se pueden soportar sobre la base de tres primitivas básicas [6]:

- *Materialización de versiones*: $Mat(Q, V_i)$, devuelve los resultados que satisfacen la consulta SPARQL Q en la versión V_i. Por ejemplo, en el caso anterior, la consulta $Mat($`(ex:Atletico,ex:hasCoach,?x)`$, V_2)$ obtendría que `ex:Manzano` era el entrenador del Atlético de Madrid en la versión V_2.

[5] Nótese que es posible anotar un mismo triple con diferentes etiquetas (versiones).

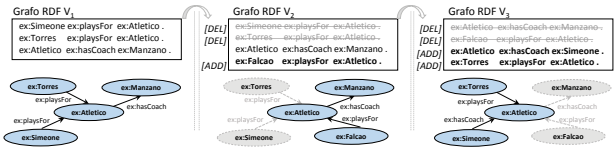


Figura 1. Ejemplo de versiones de un grafo RDF.

- *Materialización de diferencias (Delta)*: $Diff(Q, V_i, V_j)$, devuelve como resultado los valores que satisfacen Q en V_i, pero que no aparecen en V_j (*resultados borrados*), y viceversa (*resultados añadidos*). Por ejemplo, la consulta $Diff($`(?x,playsFor,ex:Atletico)`$, V_1, V_2)$ devolvería `ex:Simeone` y `ex:Torres` como resultados borrados en V_2, y `ex:Falcao` como un resultado añadido, en la misma versión.
- *Consulta de versiones*: $Ver(Q)$, resuelve Q sobre el archivo completo y anota los resultados con las versiones en las aparecen los valores recuperados. Por ejemplo $Ver($`(ex:Atletico,hasCoach,?x)`$)$ obtendría que `ex:Manzano` es un resultado válido en V_1 y V_2, mientras que `ex:Simeone` lo sería en V_3.

Estas primitivas pueden combinarse (con la semántica definida en SPARQL) para elaborar consultas más complejas. Por ejemplo, conocer qué jugadores de un equipo han sido también entrenadores se podría resolver mediante i) $Mat($`(?x,ex:playsFor,ex:Atletico)`$, V_i) \bowtie Mat($`(ex:Atletico,ex:hasCoach,?x)`$, V_j)$, $\forall i, j \in N$; o ii) realizando un join de dos consultas de versiones: $Ver($`(?x, ex:playsFor,ex:Atletico)`$) \bowtie Ver($`(ex:Atletico,ex:hasCoach,?x)`$)$.

2.1. Estado del Arte

En la actualidad, existen tres estrategias principales para representar archivos RDF [6]. Una primera aproximación consiste en mantener *copias independientes* (Ic, *independent copies*) de manera que cada versión se trata como un colección RDF independiente [13]. Esta representación conlleva un importante sobrecoste en espacio, ya que los triples que aparecen en dos o más versiones se almacenan múltiples veces. A cambio, las consultas que sólo afectan a una versión se resuelven eficientemente, mientras que el resto de primitivas (como la diferencia entre versiones) resultan altamente penalizadas.

Una aproximación totalmente contrapuesta, *basada en cambios* (Cb, *Change-based*), representa las diferencias entre dos versiones consecutivas. Es decir, la versión V_i se representa de acuerdo a sus diferencias (triples añadidos y borrados) respecto a la versión anterior $V_{(i-1)}$. Esta reorganización permite reducir notablemente el espacio requerido y favorece a aquellas consultas que operan sobre las diferencias entre versiones. Sin embargo, las consultas que requieren materializar una versión se ven afectadas negativamente de forma significativa ya que es necesario propagar todos los cambios producidos hasta la versión actualmente consultada. Para minimizar este problema, se suele optar por almacenar una

versión completa cada k deltas [5], lo que se traduce en un aumento moderado de los requisitos de almacenamiento.

Por último, las aproximaciones *basadas en marcas de tiempo* (TB, *timestamp-based*) [18] consideran una única colección RDF que contiene el conjunto de todos los triples diferentes que aparecen en alguna versión. Para cada uno de ellos, se almacena una marca de versión (habitualmente, cuándo fueron añadidos o borrados). Esta representación favorece la resolución de consultas de versiones pero, en la práctica, requiere de índices adicionales, sobre la información de las marcas, que permitan acceder a los contenidos de una versión de manera eficiente. Obviamente, estos índices adicionales también tienen un impacto no despreciable en los requisitos del almacenamiento.

Recientemente se ha publicado un trabajo sobre evaluación de archivos RDF en el que se estudian cada una de las estrategias anteriores [8]. Este estudio muestra cómo cada una de las opciones destaca en alguna de las funcionalidades requeridas, pero todas ellas tienen un importante sobrecoste en espacio. En términos cuantitativos, las soluciones estudiadas requieren entre 200 y 350GB de almacenamiento para representar un archivo RDF cuyo tamaño en gzip es de, apenas, 23GB. Atendiendo a estos resultados, es indudable que la gestión de archivos RDF requiere utilizar alguna forma de compresión.

2.2. Compresión RDF

HDT [7] fue el precursor de los compresores RDF basados en estructuras de datos compactas. Su aproximación consiste en transformar el grafo RDF en un conjunto de árboles que organizan, para cada sujeto, todos los pares (predicado, objeto) con los que se relaciona. Estos árboles se codifican utilizando secuencias de bits que proporcionan operaciones `rank/select` [10], mediante las cuales podemos navegar las ramas del árbol y resolver patrones de triples SPARQL en un espacio comprimido. Por otra parte, k^2-triples [1] se centra en aprovechar las redundancias estructurales existentes en RDF, para mejorar la efectividad de HDT. Para ello, transforma el grafo RDF en un conjunto de matrices de adyacencia (sujeto, objeto), una por cada predicado en la colección, y las comprime utilizando k^2-trees [3] (aprovechando que dichas matrices son muy poco densas). Por último, RDFCSA [2] emplea *arrays de sufijos* comprimidos e indexa los triples como *strings* cíclicos. Aunque RDFCSA no genera las representaciones más comprimidas, su rendimiento es muy competitivo y, además, sus resultados son muy estables y predecibles para todos los tipos de consultas[6]. Por este motivo, en este trabajo elegimos RDFCSA y extendemos su funcionalidad para representar y consultar archivos RDF.

3. Autoindexación de Archivos RDF (`v-RDFCSA`)

El diseño de `v-RDFCSA` parte de la experiencia adquirida por el uso de técnicas ya existentes en el estado del arte. Más concretamente, el diseño de `v-RDFCSA` se

[6] El rendimiento de HDT y k^2-triples depende del número de variables en la consulta y de su localización en el patrón, así como de la estructura de la colección de datos.

materializa como una aproximación TB *ligera* que codifica independientemente i) el conjunto de triples diferentes del archivo RDF, y ii) la información relativa a las versiones en las que se utiliza cada uno de ellos. Los triples se comprimen utilizando un autoíndice y las versiones se codifican usando representaciones sucintas de secuencias de bits. Los algoritmos de consulta diseñados en `v-RDFCSA` explotan las capacidades de autoindexación de RDFCSA [2] para recuperar los triples comprimidos y después realizan operaciones, orientadas a bit, sobre la información de versionado. Tanto los mecanismos de compresión como los algoritmos de búsqueda se explican a continuación.

3.1. Codificación de Triples RDF

`v-RDFCSA` maneja sólo el conjunto de triples diferentes utilizados en el archivo RDF. Estos *triples (sin considerar versiones)* [8] son una pequeña parte de los triples totales; por ejemplo, el archivo BEAR [8] contiene $\approx 2 \times 10^9$ triples, pero sólo ≈ 376 millones sin considerar versiones. Por ello, el problema de la codificación de triples RDF se puede reducir a la compresión tradicionalmenre realizada sobre una colección RDF habitual. En este caso, hemos elegido RDFCSA [2], un autoíndice basado en el *array de sufijos* comprimido de Sadakane (CSA) [17], que indexa un conjunto de triples en lugar de texto, y mantiene toda la funcionalidad a la hora de buscar patrones del CSA.

En primer lugar, RDFCSA transforma los triples originales usando un diccionario. Esta estructura permite reemplazar los *términos RDF* por identificadores enteros (IDs) en los rangos $[1, n_s]$ para sujetos, $[1, n_p]$ para predicados y $[1, n_o]$ para objetos. La Figura 2 (izquierda) muestra el conjunto de triples, sin considerar versiones, usados en el archivo descrito previamente. Dicho conjunto contiene $n = 5$ triples diferentes que, en la parte central, son trasformados a una representación basada en IDs[7]. El mapeo entre términos RDF e IDs se indexa independientemente usando diccionarios de texto comprimidos [15].

En la parte derecha de la figura se muestran los cuatro pasos que se realizan para convertir los triples basados en IDs en un autoíndice RDFCSA. En el primer paso, la secuencia S_{id} es una lista de ID-triples ordenados. El primer triple se almacena en $S_{id}[1, 3]$, el segundo en $S_{id}[4, 6]$, y así sucesivamente. Estos triples basados en IDs se reescriben en el paso 2 para evitar solapamientos de IDs entre sujetos, predicados y objetos. Los IDs de los sujetos no cambian y se mantienen entre $[1, n_s]$. Los IDs de predicados pasan al rango $[n_s + 1, n_s + n_p]$, y los valores para objetos a $[n_s + n_p, n_s + n_p + n_o]$. Así, el triple $(1, 1, 2)$ se transforma en $(1, 5, 8)$. Esta reasignación de IDs asegura que cada ID de sujeto es menor que cualquier ID de predicado y, a su vez, que éstos son más pequeños que cualquier ID de objeto. Esta decisión da lugar a una configuración de array de sufijos particular (paso 3) en la que los sujetos aparecen en el rango $SA[1, n]$, los predicados en $SA[n + 1, 2n]$, y los objetos en $SA[2n + 1, 3n]$. Finalmente, este array de sufijos se comprime (paso 4) usando las estructuras D y ψ de un

[7] Nótese que $n_s = 4$, $n_p = 2$, $n_o = 3$, y que los IDs 1 y 2 se usan tanto para sujetos como para objetos.

CSA[17]. $D[1,n]$ es una secuencia de bits donde los $1s$ indican el primer sufijo en SA en el que comienza cada símbolo diferente del alfabeto. A su vez, el array $\psi[1,n]$ permite recorrer el array de sufijos aprovechando que $SA[\psi[i]] = SA[i]+1$. Esto es, si $SA[i] = j$ apunta al sufijo $S[j,n]$, entonces $SA[\psi[i]] = j+1$ apunta al siguiente sufijo del texto $S[j+1,n]$. RDFCSA modifica la región $\psi[2n+1,3n]$, en la que se codifica la información de saltos desde los objetos. Originalmente, ψ permite saltar desde el objeto del k-ésimo triple al sujeto del triple $(k+1)$. Esto no es útil para resolver consultas SPARQL, ya que relaciona dos triples independientes. Nuestra decisión pasa por modificar ψ para que los valores $\psi[2n+1,3n]$ apunten al sujeto del mismo triple. Esto es, $\psi[i] \leftarrow \psi[i] - 1, \forall i \in [2n+1,3n]$ (o $\psi[i] \leftarrow n$ if $\psi[i] = 1$). De esta forma, conseguimos una codificación cíclica que comprende a los tres elementos del triple.

D y ψ permiten resolver *triple patterns* SPARQL mediante una búsqueda binaria inicial, seguida por un recorrido que recupera los triples que satisfacen el patrón buscado. En [2] se describen dichos algoritmos en profundidad.

3.2. Codificación de la Información de Versionado

La organización y codificación de los triples en RDFCSA permite que cada uno de ellos pueda identificarse fácilmente de acuerdo a la posición de su sujeto en SA. Si (s_a,p_b,o_c) es el k-ésimo triple ($1 \leq k \leq n$), podemos recuperarlo sin más que obtener su sujeto como[8] $s_a \leftarrow S[SA[k]]$, su predicado como $p_b \leftarrow S[SA[\psi[k]]]$, y su objeto mediante $o_c \leftarrow S[SA[\psi[\psi[k]]]]$.

Esta propiedad es básica para implementar las dos estrategias de codificación de la información de versionado que proponemos. Supongamos un archivo $\mathcal{A}$, que contiene N versiones diferentes y un conjunto de n *triples sin considerar versiones*. La primera estrategia de codificación (llamada `tpv`: *triples por versión*) usa N secuencias de bits $\mathcal{B}_i^v[1,n]$ para codificar los triples que aparecen en la correspondiente versión i. Esto es, si $\mathcal{B}_i^v[k] = 1$, el k-ésimo triple aparece en la versión i; en caso contrario, $\mathcal{B}_i^v[k] \leftarrow 0$. La Figura 3 (izquierda) ilustra la codificación `tpv` para el archivo de la Figura 1 (por ejemplo, la segunda versión contiene los triples 2 y 4). Nuestra segunda estrategia (llamada `vpt`: *versiones por triple*) utiliza n secuencias de bits $\mathcal{B}_i^t[1,N]$ para codificar las versiones donde aparece el k-ésimo triple. Si $\mathcal{B}_k^t$ describe el k-ésimo triple, entonces $\mathcal{B}_k^t[i] = 1$ significa que el k-ésimo triple aparece en la i-ésima versión. En caso contrario, $\mathcal{B}_k^t[i] \leftarrow 0$. La Figura 3 (derecha) muestra la codificación `vpt` para el archivo de ejemplo. Véase como el segundo triple se usa en las versiones 1 y 2.

Ambas estrategias utilizan $N * n$ bits (`tpv` incluye N secuencias de n bits cada una y `vpt` utiliza n secuencias de N bits), pero su rendimiento de búsqueda es diferente. Los algoritmos correspondientes se describen a continuación.

3.3. Algoritmos de Consulta

La explicación de los algoritmos de consulta deja de lado toda la operativa relacionada con el manejo del diccionario de *strings* y asume que tanto las

[8] $S[SA[i]] = rank_1(D,i)$, donde $rank_1$ indica el número de unos en $D[1,i]$.

Figura 2. Construcción paso a paso del RDFCSA para el archivo RDF.

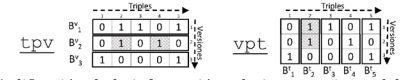


Figura 3. Codificación de la información relativa a versiones del archivo RDF.

consultas, y sus consiguientes resultados, se expresan utilizando los IDs que aparecen en S_{id} (sirva como ejemplo la Figura 2). Estas operaciones de traducción entre términos RDF e IDs se implementan utilizando primitivas de localización y extracción de *strings* en el diccionario [15].

Todos nuestros algoritmos acceden, inicialmente, a RDFCSA para recuperar el conjunto de triples candidatos que se obtiene como resultado de resolver un patrón de consulta Q' dado. Para cada triple candidato se registra la posición $SA[k]$ en la que su sujeto aparece dentro del array de sufijos. A partir de ahí, es necesario recorrer los triples candidatos y acceder a su información de versionado, con el fin de realizar las verificaciones necesarias para resolver cada consulta: si el k-ésimo triple aparece en la versión i ($Mat(Q', i)$); si un triple cambia de la versión i a la j ($Diff(Q', i, j)$); o para recuperar todas las versiones en las que aparece un triple dado ($Ver(Q')$). Nótese que los sujetos de todos los triples candidatos para las operaciones `(s??)`, `(sp?)` y `(spo)` dan lugar a un rango continuo $SA[l, r]$ en el array de sufijos[2]. Esto nos permite optimizar el acceso a la información de versiones durante el recorrido de los triples candidatos.

Consultas de materialización de versiones: $Mat(Q', i)$, recupera los triples que satisfacen Q' para una versión dada i. Una vez que las posiciones de los sujetos se han recuperado de RDFCSA, se usa la información de versionado para descartar los triples que no aparencen en la versión i. En `vpt`, cada triple candidato k se verifica mediante un acceso a la secuencia de bits $\mathcal{B}^t_k$. Si $\mathcal{B}^t_k[i] = 1$ se devuelve dicho triple, descartándolo en caso contrario. En `tpv` el proceso es similar: si $\mathcal{B}^v_i[k] = 1$, recuperamos el triple apuntado desde $SA[k]$; en caso contrario también se descarta. Los patrones `(s??)`, `(sp?)` y `(spo)` pueden optimizarse en `tpv`, ya que los sujetos de los triples candidatos son contiguos en $SA[l, r]$. Dado $c_l \leftarrow rank_1(\mathcal{B}^v_i, l)$ y $c_r \leftarrow rank_1(\mathcal{B}^v_i, r)$, el número de triples activos en la versión i dentro del rango $[l, r]$ es $c = c_r - c_l + 1$. Por lo tanto, para resolver $Mat(Q', i)$ simplemente recuperamos los triples en las posiciones $k \leftarrow select_1(\mathcal{B}^v_i, j), \forall j \in [c_l, c_r]$[9].

[9] $p \leftarrow select_1(B, j)$, indica la posición p del j-ésimo uno una secuencia de bits B.

Consultas de materialización de diferencias: $Diff(Q', i, j)$, busca los triples que satisfacen Q' y que hayan sido añadidos o borrados entre las versiones i y j. Al igual que en el caso anterior, el algoritmo recupera de RDFCSA todos las triples candidatos para Q' y, después, procesa la información de versiones para cada uno de ellos (cuyo sujeto es apuntado desde $SA[k]$). En `vpt`, esto implica dos accesos a $\mathcal{B}_k^t$: se recuperan los valores de los bits: $x \leftarrow \mathcal{B}_k^t[i]$ e $y \leftarrow \mathcal{B}_k^t[j]$, que indican si el triple k aparece en las versiones i y j, respectivamente. Análogamente, en `tpv`, se recuperan los valores $x \leftarrow \mathcal{B}_i^v[k]$ e $y \leftarrow \mathcal{B}_j^v[k]$.

A continuación, se usan los operadores $\overline{\texttt{xor}}$ y $\overline{\texttt{and}}$ para verificar cada triple candidato: *i)* si $(0 \neq ((x \ \overline{\texttt{xor}} \ y) \ \overline{\texttt{and}} \ x))$ el triple estaba activo en la versión i, pero se eliminó en la versión j. Por ello, se devuelve como un triple *eliminado*. *ii)* si $(0 \neq ((x \ \overline{\texttt{xor}} \ y) \ \overline{\texttt{and}} \ y))$ el triple no estaba en la versión i pero sí aparece en la versión j (se devuelve como un triple *añadido*). *iii)* En otro caso, el triple se descarta.

En lugar de recorrer todas los triples candidatos, en `tpv` podemos optimizar este proceso cuando dichos triples son adyacentes en $SA[l, r]$. Dada la secuencia de bits B, sea $getNext_1(B, pos)$ una función que devuelve pos si $B[pos] = 1$; y en caso contrario, devuelve la posición del siguiente 1 después de pos en B mediante $select_1(B, 1 + rank_1(B, pos))$. Dado el rango $[l, r]$, y usando $getNext_1$, podemos resolver $Diff(Q', i, j)$ en `tpv` como sigue. Calculamos $p_1 \leftarrow getNext_1(\mathcal{B}_i^v)$ y $p_2 \leftarrow getNext_1(\mathcal{B}_j^v)$. A continuación, recorremos en paralelo $\mathcal{B}_i^v$ y $\mathcal{B}_j^v$ en las posiciones que almacenan *unos*: p_1 y p_2, mientras se cumpla que $((p1 \leq r)$ *or* $(p2 \leq r))$. En cada iteración chequeamos: *i)* si $(p_1 < p_2)$, entonces sabemos que el triple en la posición p_1 fue eliminado en la versión j, y avanzamos $p_1 \leftarrow getNext_1(\mathcal{B}_i^v, p_1)$; *ii)* si $(p_2 < p_1)$ entonces el triple en la posición p_2 fue añadido en la versión j, y avanzamos $p_2 \leftarrow getNext_1(\mathcal{B}_j^v, p_2)$; *iii)* en caso contrario, calculamos $p_1 \leftarrow getNext_1(\mathcal{B}_i^v, p_1)$, $p_2 \leftarrow getNext_1(\mathcal{B}_j^v, p_2)$, y continuamos con la siguiente iteración.

Consultas de versiones: $Ver(Q')$, recupera todos los triples que satisfacen Q' y, para todos ellos, devuelve la lista de versiones en las que estaban activos. Por tanto, para cada triple candidato k, se verifica si aparece en la i-ésima versión. Esto es, $\forall i \in [1, N]$, comprobamos si el bit $\mathcal{B}_k^t[i]$ es 1, en la estrategia `vpt`, o si el bit $\mathcal{B}_i^v[k]$ es 1 en `tpv`. De nuevo es posible realizar optimizaciones en los patrones `(s??)`, `(sp?)` y `(spo)` en `tpv`. Para ello, implementamos un algoritmo de bucle anidado que recorre las N secuencias de bits $\mathcal{B}_i^v, i \in [1, N]$ y, para cada una de ellas, ejecuta un bucle interno que parte con $p \leftarrow l$ y se detiene cuando $p > r$. En cada paso del bucle devuelve como resultado el triple $p \leftarrow getNext_1(\mathcal{B}_i^v, p)$ que estaba activo en la versión i.

4. Experimentación

En esta sección presentamos los resultados de la experimentación realizada con el *benchmark* BEAR [8]. BEAR contiene un archivo RDF de evaluación que consta de 58 versiones cuyos contenidos proceden de diversos dominios. En total,

este archivo contiene $|\mathcal{A}| = 2{,}073$ millones de triples, de los cuales 376 millones son triples únicos y $3,5$ millones son triples que aparecen en todas las versiones. En promedio, el $31\,\%$ de los triples cambian entre dos versiones consecutivas, mientras que el tamaño de las versiones se incrementa desde ≈ 33 millones de triples en $|V_0|$ a ≈ 66 millones en $|V_{57}|$. El tamaño del archivo *en plano* (en formato *NTriples*) es de `325GB`, mientras que comprimido con *gzip*, ocupa `23GB`.

Además del archivo de referencia, BEAR también proporciona un conjunto variado de consultas Mat, $Diff$, y Ver. Para una comparación justa, las consultas consideradas devuelven un número similar de resultados en cada versión, y se clasifican en Q_L (bajo número de resultados), y Q_H (alto número de resultados). Para cada una de ellas, BEAR provee búsquedas por sujeto: `(s??)`, predicado: `(?p?)`, y objeto: `(??o)`. Es decir, el conjunto total de consultas comprende seis escenarios diferentes de evaluación: Q_L^S, Q_L^P, Q_L^O (consultas con baja cardinalidad) y Q_H^S, Q_H^P, Q_H^O (consultas con alta cardinalidad) para búsquedas por sujeto, predicado y objeto, respectivamente. BEAR facilita 50 consultas diferentes para cada escenario, excepto para predicados(6 y 10 consultas en Q_L^P y Q_H^P, respectivamente).

Por último, BEAR implementa un sistema de referencia para manejar archivos RDF basado en Jena TDB[10]. Este sistema considera tres variantes de indexación para reflejar las tres principales estrategias de representación descritas previamente: i) Jena-IC indexa cada versión en un repositorio independiente; ii) Jena-CB crea dos índices por cada versión, para los triples añadidos y eliminados; y iii) Jena-TB usa el nombre del grafo para anotar cuando se añade o borra cada triple, empleando un único repositorio TDB.

Nuestra implementación de `v-RDFCSA` está realizada en lenguaje C y considera dos variantes, correspondientes a las estrategias `vpt` y `tpv`. En ambos casos, evaluamos distintos valores de muestreo para la estructura ψ en RDFCSA ($t_\psi = \{16, 64, 256\}$). Además, consideramos estructuras de bits planas y comprimidas con RRR [16]. En este segundo caso, concatenamos todas las secuencias de bits `vpt`, y aplicamos RRR sobre la secuencia de bits resultante, con el objetivo de mejorar los requisitos de almacenamiento de la solución. Finalmente, destacar también que nuestra variante `tpv-RRR-OPT` explota las particularidades de RRR a la hora de resolver consultas por sujeto.

El estudio experimental ha sido realizado en una máquina Intel Xeon E5-2650v2 @ 2.60GHz (32 núcleos), 256GB RAM,Debian 7.8, y nuestros prototipos se han compilado con `gcc 4.7.2` (opción `-O9`).

Tradeoffs espacio/tiempo. En primer lugar, estudiamos el rendimiento de `v-RDFCSA` en todas sus variantes. Por motivos de espacio, mostramos únicamente los resultados más representativos. En este caso, optamos por aquellos escenarios en los que se obtienen un gran número de resultados (alta cardinalidad), ya que las conclusiones son similares para aquellas consultas con menor número de resultados. Además, descartamos las consultas por predicado ya que

[10] https://jena.apache.org/documentation/tdb/

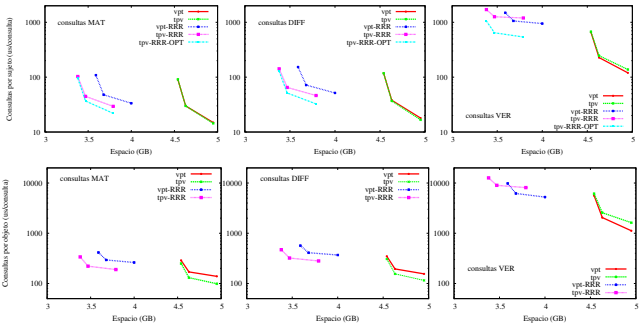


Figura 4. Comparación espacio/tiempo. Búsquedas por sujeto (arriba) y por objeto (abajo): consultas Mat, $Diff$, y Ver.

su rendimiento se enmarca entre el obtenido por las búsquedas por sujetos y por objetos.

La Figura 4 muestra los *tradeoffs* espacio/tiempo obtenidos por `v-RDFCSA`. El eje X representa los requisitos de espacio (en GB), mientras que el eje Y presenta tiempos de consulta (en μs por consulta). Centrándonos en los requisitos de espacio, las variantes en plano de `vpt` y `tpv` necesitan $4,54 - 4,95$GB (el espacio se incrementa de $t_\psi = 256$ a $t_\psi = 16$), pero el uso de RRR lo reduce a $3,38 - 3,79$GB (`tpv`) y $3{,}59 - 4{,}00$GB (`vpt`). Este hecho demuestra que la información de versionado es muy compresible, al igual que el propio RDF. En este caso, utilizar secuencias de bits comprimidas puede ahorrar más de 1GB. Sin embargo, RRR es más lento que la variante que emplea secuencias de bits en plano. No obstante, aunque la diferencia es importante en las consultas Ver, ambas alternativas compiten en las consultas Mat y $Diff$. La comparación entre `vpt` y `tpv` muestra que ambas se comportan de manera muy similar en las consultas por sujeto, debido a que ambas configuraciones pueden aprovechar la localidad en el acceso a las secuencias de bits. La diferencia se incrementa en las búsquedas por objeto, debido a que los posibles resultados se encuentran dispersos en el array de sufijos y, por tanto, se producen más fallos de caché al acceder a las secuencias de bits. Este hecho revela qué configuración (de secuencia de bits) es mejor para cada tipo de consulta. Por una parte, `tpv` es la opción más rápida para las consultas Mat y $Diff$, ya que para resolverlas sólo revisa una o dos versiones, respectivamente. Por otra parte, `vpt` es la opción preferida para Ver, considerando que para resolver este tipo de consultas es necesario revisar todas las versiones de los triples recuperados de RDFCSA.

Para las consultas por sujeto, `tpv` tarda, en promedio, 14-91μs y 17-118μs, para cada operación, Mat y $Diff$, respectivamente; mientras que `tpv-RRR-OPT` necesita 22-95μs y 32-128μs, en los mismos casos. En las consultas Ver, `vpt` pre-

cisa, de media, 120-660μs, frente a 539-1055μs, en `tpv-RRR-OPT`. El rendimiento obtenido en las consultas por objeto es menos competitivo debido a que su resolución es más compleja en RDFCSA. En este caso, la variante `tpv` consigue una media de 99-250μs y 115-307μs (para las operaciones Mat y $Diff$); mientras que `tpv-RRR` reporta 189-338μs y 281-470μs, respectivamente. En las consultas Ver, `vpt` necesita $1,1$-$5,6ms$ por consulta, frente a $5,2$-$9,8ms$ en `vpt-RRR`.

Comparación con el Sistema de Referencia. A continuación, comparamos las variantes más competitivas de `v-RDFCSA` (con $t_\psi = 64$) respecto al sistema de referencia propuesto en BEAR. Para una comparación más justa, integramos un diccionario *Front-Coding* [15] que permite transformar en *términos RDF* los resultados recuperados directamente desde nuestra propuesta. De este modo, nuestro conjunto de resultados es el mismo que el obtenido por la implementación del sistema de referencia usando Jena. Este diccionario añade 2,3GB extra al espacio requerido por `v-RDFCSA`. Además, consideramos el tiempo de transformación de los resultados a términos RDF, lo que supone una pequeña sobrecarga respecto a los resultados presentados en la sección anterior.

`v-RDFCSA` mejora notablemente el espacio obtenido por las diferentes estrategias implementadas en el sistema de referencia. Nuestras variantes emplean $\approx$ 5,7-7,3 GB, mientras que Jena-IC, Jena-CB, y Jena-TB necesitan 225GB, 196GB, y 353GB, respectivamente. Esta mejora en espacio se complementa con una resolución de consultas notablemente más eficiente. La Figura 5 compara el tiempo de resolución para consultas por sujeto y objeto, con alta cardinalidad. La gráfica Mat (izquierda) muestra el tiempo medio de resolución (eje Y) para cada versión en el archivo RDF (eje X), mientras que $Diff$ (centro) muestra el tiempo medio para consultas que computan las diferencias entre la versión inicial e intervalos crecientes de 5 versiones (como se define en BEAR). Para una mayor claridad, sólo incluimos los resultados de `v-RDFCSA` para `tpv` y `tpv-RRR(-OPT)`, ya que el rendimiento del resto de alternativas está en el mismo orden de magnitud. Por último, la gráfica derecha presenta el tiempo medio de resolución para las consultas Ver, considerando las variantes `vpt` en `v-RDFCSA`.

Dentro del sistema de referencia, Jena-IC se postula como la mejor alternativa, aunque su rendimiento es similar a Jena-TB para las consultas Ver. Sin embargo, nuestras variantes mejoran todas las estrategias consideradas en el sistema de referencia para consultas Mat y $Diff$ en búsquedas por sujeto y objeto. La diferencia a nuestro favor es de, aproximadamente, un orden de magnitud. Jena-IC obtiene tiempos en el orden de 1-4ms/consulta, mientras que `v-RDFCSA` necesita $0,1$-$0,4ms$/consulta. En el caso de las consultas Ver, el resultado es similar, si bien nuestra ventaja se ve reducida ligeramente en las búsquedas por objeto. Nuestra mejor alternativa (`vpt`) obtiene $0,5ms$/consulta y $3,5ms$/consulta para búsquedas por sujeto y objeto, respectivamente, frente a $61,5ms$/consulta and $73,5ms$/consulta requerido por Jena-TB.

En conclusión, `v-RDFCSA` demuestra cómo la gestión de archivos RDF comprimidos no sólo reduce el espacio requerido (mejorando la escalabilidad de las representaciones), sino que también mejora los tiempos de resolución de las consultas más relevantes en este escenario.

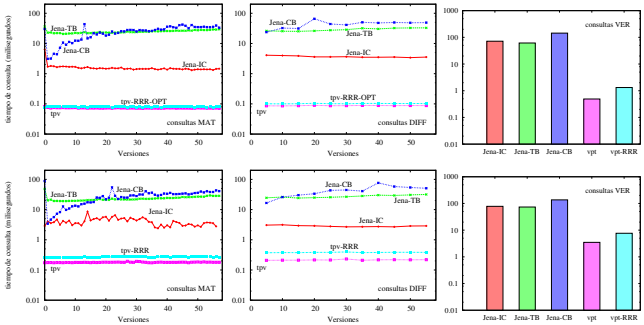


Figura 5. Comparación con el marco de referencia. Búsquedas por sujeto (arriba) y por objeto (abajo): consultas Mat, $Diff$, y Ver.

5. Conclusiones y Trabajo Futuro

En este artículo hemos presentado `v-RDFCSA`, la primera técnica diseñada específicamente para la gestión y consulta de archivos RDF comprimidos. Nuestra propuesta se ha construido como una extensión del autoíndice RDFCSA y considera dos estrategias para la codificación de la información de versionado asociada a cada triple en el archivo. Ambas estrategias están basadas en estructuras de bits con soporte para operaciones `rank` y `select`. Esta decisión no sólo garantiza un espacio de almacenamiento reducido, sino también la resolución eficiente de consultas en memoria principal.

Nuestra propuesta ha sido validada experimentalmente utilizando el *benchmark* BEAR, obteniendo unos resultados muy competitivos tanto en espacio de almacenamiento como en rendimiento de consulta. Las variantes de `v-RDFCSA` necesitan hasta 60 veces menos espacio de almacenamiento que las técnicas consideradas en el sistema de referencia y mejoran su rendimiento de consulta hasta en un orden de magnitud.

Estos resultados asientan nuestras expectativas en esta nueva línea de investigación y abren nuevos retos futuros. Actualmente, estamos trabajando en la implementación de algoritmos adicionales para la resolución de consultas SPARQL avanzadas con las que aumentar la funcionalidad provista por `v-RDFCSA`. Complementariamente, estamos extendiendo HDT para evaluar su rendimiento en este escenario. De acuerdo a nuestros experimentos iniciales, se espera que la solución basada en HDT sea algo menos competitiva en espacio, pero con un rendimiento de consulta altamente competitivo.

Agradecimientos. La investigación presentada en este artículo ha sido financiada por los proyectos TIN2013-46238-C4-3-R, TIN2013-47090-C3-3-P y

TIN2015-69951-R del MINECO (PGE y FEDER); proyecto ITC-20151247 CDTI,MINECO; ICT COST Action IC1302; proyecto GRC2013/053 de la Xunta de Galicia (FEDER) y Austrian Science Fund (FWF): M1720-G11.

Referencias

[1] S. Álvarez-García, N. Brisaboa, J.D. Fernández, M.A. Martínez-Prieto, and G. Navarro. Compressed Vertical Partitioning for Efficient RDF Management. *Knowl. Inf. Syst.*, 44(2):439–474, 2015.

[2] N. Brisaboa, A. Cerdeira, A. Fariña, and G. Navarro. A compact RDF store using suffix arrays. In *Proc. of SPIRE*, pages 103–115, 2015.

[3] N. Brisaboa, S. Ladra, and G. Navarro. Compact Representation of Web Graphs with Extended Functionality. *Infor. Syst.*, 39(1):152–174, 2014.

[4] F. Claude, A. Fariña, M.A. Martínez-Prieto, and G. Navarro. Universal Indexes for Highly Repetitive Document Collections. *Information Systems*, 2016. Disponible en `http://www.dcc.uchile.cl/ gnavarro/ps/is16.3.pdf`.

[5] I. Dong-Hyuk, L. Sang-Won, and K. Hyoung-Joo. A Version Management Framework for RDF Triple Stores. *Int. J. Softw. Eng. Know.*, 22(1):85–106, 2012.

[6] J. D. Fernández, A. Polleres, and J. Umbrich. Towards Efficient Archiving of Dynamic Linked Open Data. In *Proc. of DIACHRON*, pages 34–49, 2015.

[7] J.D. Fernández, M.A. Martínez-Prieto, C. Gutiérrez, A. Polleres, and M. Arias. Binary RDF Representation for Publication and Exchange. *J. Web Semant.*, 19:22–41, 2013.

[8] J.D. Fernández, J. Umbrich, and A. Polleres. BEAR: Benchmarking the Efficiency of RDF Archiving. Technical report, 2015. Disponible en `http://epub.wu.ac.at/4615/`.

[9] D. Gomes, M. Costa, D. Cruz, J. Miranda, and S. Fontes. Creating a Billion-scale Searchable Web Archive. In *Proc. of WWW Companion*, pages 1059–1066, 2013.

[10] R. González, S. Grabowski, V. Mäkinen, and G. Navarro. Practical implementation of rank and select queries. In *Proc. of WEA*, pages 27–38, 2005.

[11] S. Harris and A. Seaborne. *SPARQL 1.1 Query Language*. W3C Recomm., 2013. `http://www.w3.org/TR/sparql11-query/`.

[12] T. Käfer, A. Abdelrahman, J. Umbrich, P. O'Byrne, and A. Hogan. Observing Linked Data Dynamics. In *Proc. of ISWC*, pages 213–227, 2013.

[13] M. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov. Ontology Versioning and Change Detection on the Web. In *Proc. of EKAW*, pages 197–212, 2002.

[14] F. Manola and E. Miller. *RDF Primer*. W3C Recomm., 2004. `www.w3.org/TR/rdf-primer/`.

[15] M.A. Martínez-Prieto, N. Brisaboa, R. Cánovas, F. Claude, and G. Navarro. Practical Compressed String Dictionaries. *Infor. Syst.*, 56:73–108, 2016.

[16] R. Raman, V. Raman, and S. Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proc. of SODA*, pages 233–242, 2002.

[17] K. Sadakane. New Text Indexing Functionalities of the Compressed Suffix Arrays. *J. Algorithm*, 48(2):294–313, 2003.

[18] M. Vander Sander, P. Colpaert, R. Verborgh, S. Coppens, E. Mannens, and R. Van de Walle. R&Wbase: Git for Triples. In *Proc. of LDOW*, 2013. CEUR-WS 996, paper 1.

Compresión de Big Semantic Data basada en HDT y MapReduce

José M. Giménez-García[1], Javier D. Fernández[2], and Miguel A. Martínez-Prieto[3]

[1] Univ Lyon, UJM-Saint-Etienne, CNRS, Laboratoire Hubert Curien UMR 5516, F-42023 Saint Etienne (Francia)
jose.gimenez.garcia@univ-st-etienne.fr,
[2] Vienna University of Economics and Business (Austria)
jfernand@wu.ac.at
[3] DataWeb Research, Departmento de Informática, Univ. de Valladolid (España)
migumar2@infor.uva.es

Resumen HDT es un formato binario que nace con el objetivo de reducir los requisitos de almacenamiento que presentan las sintaxis RDF tradicionales. Su estructura interna no sólo favorece la compresión, sino que también facilita la resolución de algunas consultas de interés en el ámbito de la Web Semántica. Sin embargo, el proceso de codificación HDT requiere una cantidad importante de memoria principal, lo que limita su adopción en aplicaciones que manejan *Big Semantic Data*. Para afrontar este problema, proponemos HDT-MR, una revisión del algoritmo de construcción de HDT basada en tecnología MapReduce. Los experimentos que presentamos en este artículo demuestran que el coste de HDT-MR crece linealmente con el volumen de los datos de entrada, lo cual facilita la serialización de colecciones RDF de miles de millones de triples utilizando un pequeño *cluster Hadoop*.

Keywords: Web Semántica, Compresión, RDF, HDT, MapReduce

1. Introducción

RDF (*Resource Description Framework*) [8] se ha convertido en el formato de referencia para la publicación e intercambio de datos en la Web de Datos. Iniciativas como *Linked Open Data* ponen de manifiesto su potencial para la integración de conjuntos heterogéneos de datos, con diferentes grados de estructura y procedentes de fuentes diversas. La flexibilidad de RDF se debe a la estructura ternaria (*triple*) que utiliza para describir la información: (i) el *sujeto* identifica el recurso que está siendo descrito, (ii) el *predicado* establece una propiedad sobre dicho recurso y (iii) el objeto fija el valor de la propiedad para el recurso descrito. Una colección RDF está formada por un conjunto de triples cuyos sujetos y objetos pueden verse como nodos de un grafo dirigido y en el que los predicados juegan el rol de aristas. Aunque esta noción de grafo es una buena metáfora para entender la forma en la que RDF organiza la información, el almacenamiento

y/o intercambio de estas colecciones requiere el uso de formatos de serialización. Esta necesidad ha sido reconocida explícitamente en la última propuesta del *RDF Primer*[4], realizada por el grupo de trabajo en RDF dentro del *World Wide Web Consortium (W3C)*. Este documento sugiere diversas alternativas (JSON-LD, RDF/XML o los formatos basados en Turtle) que serializan el grafo RDF en *texto plano*. Esta decisión simplifica, notablemente, el proceso de escritura de los triples y puede implementarse sin grandes sobrecargas computacionales. En contrapartida, los ficheros resultantes tienden a ser muy voluminosos. Una solución trivial (y muy usada en la práctica) es utilizar un compresor universal, como `gzip`, para reducir el tamaño de estos ficheros. Sin embargo, esta decisión sólo oculta el problema, ya que el uso efectivo de los triples requiere un proceso previo de descompresión que devuelve la colección a su tamaño original.

HDT (*Header-Dictionary-Triples*) es un formato binario de serialización que surge como alternativa a las propuestas anteriores. HDT codifica la colección utilizando dos componentes: *Diccionario* y *Triples*. El *Diccionario* asocia cada término usado en el grafo (URIs y literales) con un identificador numérico único (ID). Esta decisión permite obtener un grafo de IDs, cuya codificación binaria se realiza en el componente *Triples*. Ambos componentes se serializan en espacio comprimido y permiten acceder a los datos sin necesidad de descomprimirlos previamente [9]. Esta propiedad ha facilitado que HDT pase a considerarse una pieza básica para la construcción de motores de almacenamiento semánticos. *HDT-FoQ* [9] muestra el potencial de la tecnología HDT para la resolución de algunas consultas SPARQL, mientras que *WaterFowl* [3] incopora HDT en su motor de inferencia. Otras tecnologías como *Linked Data Fragments* [13] o el sistema de recomendación *SemStim* demuestran la eficiencia de HDT como motor de almacenamiento.

Todos los beneficios generados por HDT, en el usuario final, están sujetos al coste de serialización que debe pagar el publicador de la colección. La construcción de los componentes *Diccionario* y *Triples* requiere un procesamiento exhaustivo de toda la colección RDF para el que necesita un equipo de cómputo con una gran cantidad de memoria RAM. Por lo tanto, estamos ante un proceso poco escalable que dificulta la serialización de colecciones RDF de gran tamaño (en el orden de los cientos o miles de millones de triples). Este tipo de colecciones no son muy comunes (aunque comienzan a aparecer en áreas como la biología o la astronomía), pero sí es más habitual encontrarnos con *mashups* que integran datos heterogéneos procedentes de fuentes diversas (por ejemplo, en ámbitos como las *smart cities*) y que materializan la noción de *Big Semantic Data*.

En este artículo proponemos HDT-MR, una nueva implementación del *workflow* de publicación de colecciones RDF (basado en HDT) desarrollada sobre el *framework* MapReduce [4]. Esta decisión supone una mejora sustancial en la escalabilidad del proceso original de construcción de HDT y nos permite serializar colecciones RDF de gran tamaño. En los experimentos actuales, HDT-MR procesa colecciones 10 veces más grandes que las serializadas con la propuesta original y obtiene tiempos de codificación que crecen linealmente con el tamaño

[4] `http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624`

de los datos de entrada. Estos resultados avalan la integración de HDT-MR como componente principal del *workflow* de serialización HDT, garantizando la escalabilidad del mismo y preservando todas las características originales de formato de cara a su explotación en el usuario final [9,5].

El resto artículo se organiza como sigue: en la sección 2 se presentan los conocimientos necesarios para entender la propuesta actual, cuya descripción se realiza en la sección 3. La sección 4 describe los resultados experimentales obtenidos con HDT-MR y, finalmente, la sección 5 discute nuestras conclusiones actuales y las líneas de trabajo futuro que surgen entorno a ellas.

2. Background

Esta sección resume los fundamentos de MapReduce y, a continuación, describe la tecnología HDT y su relevancia en el ámbito de la compresión RDF.

2.1. MapReduce

MapReduce [4] plantea un modelo de programación y un *framework* optimizado para procesar grandes volúmenes de datos en entornos distribuidos. Su objetivo principal es abstraer la complejidad subyacente a estos entornos y ofrecer un mecanismo eficiente que permita paralelizar el procesamiento de datos de cualquier naturaleza. Una tarea (*job*) MapReduce comprende dos fases. La primera fase (`map`) lee los datos de entrada como pares clave-valor $(k1, v1)$ y genera, como resultado, series de pares clave-valor en un dominio diferente al inicial $(k2, v2)$. La segunda fase (`reduce`) procesa los valores $v2$, relacionados con cada clave $k2$, y obtiene una lista final de resultados. Cada una de las fases ejecuta múltiples procesos que actúan sobre una pequeña parte de los datos de entrada y, por tanto, generan una parte del resultado final.

MapReduce implementa una arquitectura maestro-esclavo (*master-slave*). El nodo maestro se encarga de iniciar la tarea, distribuir la carga de trabajo dentro del *cluster* y procesar la información administrativa generada durante la ejecución, mientras que los nodos esclavos (trabajadores) ejecutan los `map` y/o `reduce` asignados. MapReduce hace un uso exhaustivo de operaciones de I/O. Los trabajadores leen y escriben datos en discos que implementan el modelo de ficheros distribuido GFS *(Google File System)*, de forma que los resultados temporales se almacenan en diferentes nodos del *cluster*. Esto favorece la *localidad de los datos* y maximiza el rendimiento individual de los trabajadores. Aún así, todavía hay información que debe transferirse entre los nodos.

Apache Hadoop[5] es la implementación de MapReduce más utilizada actualmente. Hadoop introduce un modelo de almacenamiento distribuido denominado HDFS *(Hadoop Distributed File System)* que permite gestionar el factor de replicación de los datos (comúnmente, cada fragmento de datos se replica en tres nodos diferentes), con el objetivo de mejorar su localidad y, a su vez, incrementar la tolerancia a la ocurrencia de fallos.

[5] `http://hadoop.apache.org/`

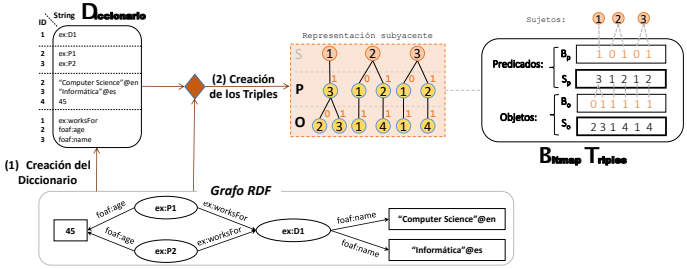


Figura 1. Componentes *Diccionario* y *Triples* (HDT) para codificar un grafo RDF.

2.2. HDT

HDT[6] [5] es un formato binario diseñado para optimizar el almacenamiento y la transmisión de ficheros RDF. HDT codifica una colección RDF utilizando tres componentes: i) la *Cabecera (Header)* contiene los metadatos necesarios para el descubrimiento y el parseo del fichero HDT; ii) el *Diccionario (Dictionary)* es un catálogo que organiza el conjunto de términos diferentes utilizados en la colección y asigna, a cada uno de ellos, un identificador único: ID; iii) finalmente, el componente *Triples* reemplaza los términos utilizados en cada triple RDF por su correspondiente ID y codifica sucintamente el grafo de IDs resultante. Los componentes *Diccionario* y *Triples* (ilustrados en la figura 1) son los responsables de los resultados de compresión que obtiene HDT.

El *Diccionario* organiza los términos utilizados en la colección de acuerdo al rol que desempeñan en la misma. Esto da lugar a cuatro particiones disjuntas: los términos que juegan el papel de sujetos y objetos (**SO**) se identifican en el rango `[1, |SO|]` (siendo `|SO|` el número de términos diferentes que desempeñan este rol); los términos que desempeñan, exclusivamente, roles de sujeto (**S**) u objeto (**O**) (ambas secciones utilizan IDs en el rango de `|SO|+1` a `|SO|+|S|` y `|SO|+|O|`, respectivamente, siendo `|S|` y `|O|` el número de sujetos y objetos exclusivos); finalmente, los predicados (**P**) se identifican en el rango `[1, |P|]`, donde `|P|` cuantifica el número total de predicados en la colección. Nótese que no existe una partición que combine los términos que aparecen como predicado y, simultáneamente, sujeto y/o objeto. Por ello, dichos términos aparecerán repetidos en la secciones que corresponda. Cada sección del *Diccionario* se codifica independientemente para aprovechar sus características particulares. La codificación de diccionarios [10] es un problema ortogonal al tratado en este trabajo, por lo tanto no profundizaremos más en el.

El componente *Triples* codifica el grafo que resulta de reemplazar los términos por sus correspondientes IDs en el *Diccionario*. Esto es, los triples RDF se codifican como tuplas de tres IDs (ID-triples a partir de aquí): $(\texttt{id}_s\ \texttt{id}_p\ \texttt{id}_o)$, donde

[6] HDT es una *Member Submission* del W3C: `http://www.w3.org/Submission/HDT/`

$\texttt{id}_s$, $\texttt{id}_p$, y $\texttt{id}_o$ son, respectivamente, los IDs de los términos sujetos, predicado y objeto en el *Diccionario*. Este componente codifica los triples como un conjunto de árboles, uno por cada sujeto en la colección: la raíz de cada árbol identifica al propio sujeto, el nivel intermedio contiene la lista ordenada de los predicados que establecen las propiedades del sujeto y, finalmente, las hojas listan los IDs de los objetos que fijan los valores de cada propiedad del sujeto. Esta organización codifica los IDs de predicados y objetos (que describen el nivel intermedio y las hojas de los árboles) mediante *dos secuencias* de números enteros: `Sp` y `So`, y *dos secuencias de bits*: `Bp` y `Bo`, alineadas con las anteriores [5].

Serialización HDT. Una vez descritos los componentes *Diccionario* y *Triples*, estamos en disposición de explicar el proceso de serialización que, actualmente, utiliza HDT. Este proceso contempla tres etapas principales.

1. Organización de los términos RDF. Esta primera etapa, procesa la colección y clasifica los términos de cada triple en su sección del *Diccionario*. Para ello, construye (en RAM) tres tablas *hash*, que implementan las relaciones *sujeto-ID*, *predicado-ID* y *objeto-ID*, y busca el sujeto, el predicado y el objeto en la tabla correspondiente. Si el término existe, se recupera el ID asociado; en caso contrario, se inserta el nuevo término en la tabla y se le asigna un ID autoincremental. Estos IDs se utilizan para obtener una representación ID-triples temporal que se almacena en un *array* (también en memoria principal). Una vez leído el fichero de entrada, se procesan las tablas que contienen los *mappings* de sujetos y objetos para identificar aquellos términos que desempeñan ambos roles en la colección. Estos términos comunes se descartan en sus respectivas estructuras y se insertan en una cuarta tabla *hash* que implementa el *mapping* correspondiente a la sección SO del diccionario.

2. Construcción del Diccionario. Cada sección del *Diccionario* se ordena lexicográficamente con el objetivo de aprovechar la existencia de prefijos comunes entre los términos y, con ello, reducir el espacio requerido para su codificación [10]. Finalmente, se construye un *array* auxiliar que almacena la permutación de los IDs desde su orden inicial (en la tabla *hash*) a su orden final.

3. Construcción de los Triples. La etapa final recorre el *array* temporal de IDs-triples y reemplaza los IDs inicales, de cada triple, por los obtenidos tras la ordenación del *Diccionario*. Una vez actualizado, el *array* de ID-triples se ordena por sujeto, predicado y objeto de cara a su codificación final. Esta proceso de codificación sólo requiere un recorrido secuencial del *array* de ID-triples en el que se extraen los pares (predicado,objeto) para cada sujeto y se almacenan en las secuencias `Sp/So`, actualizando, coordinadamente, las secuencias de bits.

2.3. Trabajo Relacionado

Atendiendo a la taxonomía presentada en [11], HDT puede considerarse un compresor *sintáctico* dada su capacidad para detectar redundancia a nivel de serialización. Por un lado, el *Diccionario* aborda la redundancia simbólica existente en los términos de la colección, mientras que el componente *Triples* explota

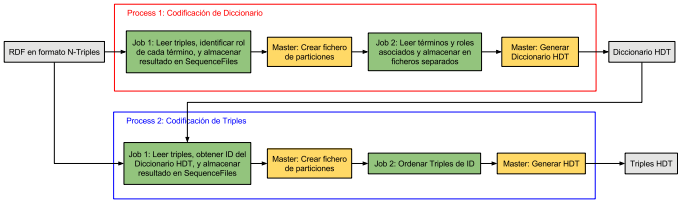


Figura 2. Descripción del *workflow* HDT-MR.

la redundancia estructural subyacente a la topología del grafo. Los compresores sintácticos son los que consiguen unos mejores resultados, en la práctica. Propuestas como k^2-triples [14] o RDFCSA [1] utilizan diferentes estrategias de codificación que reducen notablemente el tamaño de los ficheros RDF. Ambas soluciones también comparten la necesidad de llevar a cabo un proceso de compresión complejo que también requiere grandes cantidades de memoria. Por otra parte, los compresores *lógicos* se centran en detectar aquellos triples que pueden inferirse a partir de otros triples ("primitivos") y sólo codifican la parte "primitiva" de la colección original. Técnicas como la propuesta en [7] eliminan hasta más del 50 % de los triples, pero su efectividad no compite con la obtenida por los compresores sintácticos. Además, el proceso de detección de los triples redundantes también es complejo y exhaustivo en el uso de recursos. Un trabajo más reciente [11] ha presentado una propuesta que detecta y explota tanto la redundancia sintáctica como la semántica. Sus resultados mejoran ligeramente a HDT, pero quedan lejos de los obtenidos por técnicas como k^2-triples. En sintonía con todas las técnicas anteriores, su proceso de compresión resulta muy complejo en la práctica.

En resumen, los compresores RDF están lastrados por un problema importante de escalabilidad que ya ha sido estudiado en el ambiente MapReduce [12]. En este caso, se implementó un algoritmo que construía el *mapping* entre términos e IDs y, posteriormente, reescribía los triples de acuerdo a los IDs de sus términos. Un trabajo más reciente [2] ha abordado este problema mediante otro algoritmo distribuido desarrollado, en lenguaje X10. En ambos casos, los resultados son un paso adelante respecto al estado del arte actual.

3. HDT-MR

HDT-MR se implementa mediante un proceso MapReduce que comprende dos etapas centradas en la codificación de los componentes HDT (ver figura 2). Cada una de estas etapas, y sus consiguientes *jobs*, se explican a continuación.

3.1. Codificación del Diccionario

Esta primera etapa se centra en la construcción del *Diccionario* HDT, partiendo de la premisa de que la colección RDF original se encuentra en formato

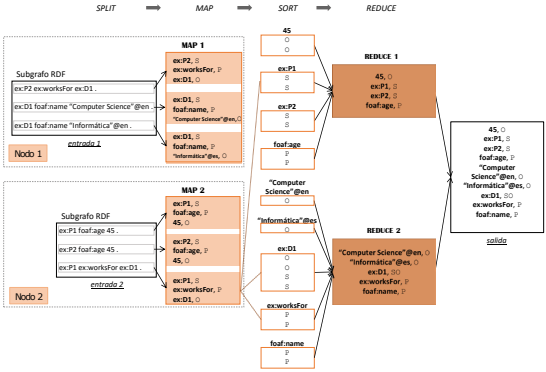


Figura 3. Job 1.1: identificación de roles.

Algoritmo 1 Job 1.1: identificación de roles.

```
function MAP(key,value)          ▷ key: line number (discarded)      ▷ value: triple
    emit(value.subject, "S")
    emit(value.predicate, "P")
    emit(value.object, "O")
end function
function COMBINE/REDUCE(key,values)      ▷ key: RDF term      ▷ value: roles (S, P, and/or O)
    for role in values do
        if role contains "S" then isSubject ← true
        else if role contains "P" then isPredicate ← true
        else if role contains ."O" then isObject ← true
        end if
    end for
    roles ← ""
    if isSubject then append(roles, "S")
    else if isPredicate then append(roles, "P")
    else if isObject then append(roles, "O")
    end if
    emit(key, roles)
end function
```

N-Triples. Para su construcción, es necesario identificar el rol de cada término en la colección, obtener las cuatro particiones (en orden lexicográfico) que describen el Diccionario y, finalmente, codificarlas. Para ello, empleamos dos *jobs* MapReduce y dos procesos locales que se ejecutan en el nodo *maestro*.

Job 1.1: Identificación de roles. Este *job* se encarga de identificar los roles que juegan los términos RDF en la colección. Para ello, los *mappers* procesan (uno a uno) los triples y emiten pares clave-valor de la forma `(término RDF, rol)`, donde el valor de rol puede ser S (sujeto), P (predicado), u O (objeto), de acuerdo a la posición del término en el triple. La figura 3 muestra un *cluster* básico de dos nodos, en el cada uno de ellos participa en el procesamiento de la colección representada en la figura 1. Por ejemplo, los pares `(ex:P1,`S`)`, `(ex:worksFor,`P`)`, y

`(ex:D1,`O`)` son los resultados de procesar el triple `(ex:P1, ex:worksFor, ex:D1)`. La salida de cada `map` pasa por un *combiner* (en cada nodo en el que se realizan las tareas `map`) antes de ser enviada a los *reducer*. Su objetivo es eliminar los pares `(término RDF, rol)` repetidos y, con ello, reducir los costes de comunicación.

Los pares clave-valor resultantes se ordenan y distribuyen entre los *reducers*, que se encargan de identificar los diferentes roles asociados a cada término. Cuando un término tiene asociados valores S y O, el *reducer* genera un par `(término RDF,` SO`)`. En caso contrario, genera como salida el par original asociado al término. El resultado del *job* consiste en varias listas de pares `(término RDF, rol)` (tantas como número de *reducers*). El algoritmo 1 muestra el pseudocódigo que implementa este *job*.

Job 1.2: Ordenación de los términos. Para construir las diferentes secciones de un *Diccionario* HDT, es necesario que todos los términos están ordenados lexicográficamente en un único fichero de entrada. Sin embargo, el *job* anterior genera múltiples ficheros, cuyos contenidos están ordenados por el valor de la clave. Además, se plantea una cuestión adicional acerca de qué términos están en cada fichero. Por defecto, Hadoop utiliza una función *hash* sobre la clave antes de asignar el par a un determinado *reducer* con el objetivo de distribuir uniformemente el universo de claves entre el conjunto de *reducers* disponibles.

No obstante, el *framework* permite configurar diferentes políticas de distribución. La alternativa más sencilla y, a la vez, menos eficiente pasa por utilizar un único *reducer* que procese todos los pares. Esta opción sería equivalente a renunciar al procesamiento paralelo que ofrece Hadoop. La segunda alternativa pasaría por crear manualmente los grupos de claves que se asignarán a cada nodo. Sin embargo, estas particiones deberían elegirse cuidadosamente ya que cualquier desequilibrio provocaría que alguno de los *reducers* necesitase más tiempo que los demás para acabar su tarea y, por consiguiente, se convertiría en el cuello del botella del proceso. Descartadas las opciones anteriores, HDT-MR usa el *TotalOrderPartitioner* para muestrear los pares de entrada de una manera más eficiente. Sin embargo, este particionamiento no puede llevarse a cabo durante la ejecución de *job*, sino que tiene que completarse antes de su inicio.

Esta situación motiva la inclusión del segundo *job*, cuyo objetivo es obtener una ordenación global de la salida en el *job* anterior. Así, se toma como entrada las listas de pares `(término RDF, rol)`, obtenidas en el *job* anterior, y se agrupan de acuerdo a su valor de rol. Para ello, empleamos *identity mappers*, que envían directamente su entrada a los *reducers* sin realizar procesamiento alguno. Por su parte, los *reducers* simplemente generan como salida una lista ordenada de los términos correspondientes a cada rol. La figura 4 ilustra este *job*. Nótese que la construcción del Diccionario HDT sólo requiere el valor del término RDF, por lo que los *reducers* emiten pares de la forma `(término RDF, null)`. El resultado final de cada *reducer* comprende cuatro listas, una por cada sección del *Diccionario*, que finalmente se concatenan para obtener el orden global de cada una de estas secciones. El pseudocódigo de este *job* se describe en el algoritmo 2.

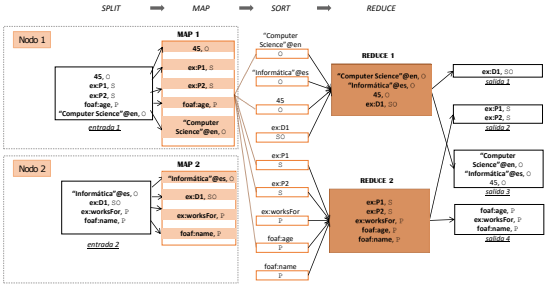


Figura 4. Job 1.2: ordenación de los términos.

Algoritmo 2 Job 1.2: ordenación de los términos.

```
function REDUCE(key,value)          ▷ key: RDF term          ▷ value: roles (S, P, and/or O)
    for resource in values do
        if resource contains 'S' then isSubject ← true
        else if resource contains 'P' then isPredicate ← true
        else if resource contains 'O' then isObject ← true
        end if
    end for
    output ← ""
    if isSubject & isObject then emit_to_SO(key, null)
    else if isSubject then emit_to_S(key, null)
    else if isPredicate then emit_to_P(key, null)
    else if isObject then emit_to_O(key, null)
    end if
end function
```

Proceso local 1.3: Codificación del Diccionario HDT La tarea final, de la etapa de codificación del Diccionario, se realiza en el nodo *maestro* y se centra, exclusivamente, en la compresión de las cuatro secciones obtenidas en el *job* anterior. Para cada una de ellas, el proceso lee (línea a línea) los términos y los codifica utilizando la técnica *Plain Front-Coding* [10]. Este es un proceso sencillo que no presenta problemas de escalabilidad.

3.2. Codificación de los Triples

Esta segunda etapa realiza una nueva lectura de la colección original con el objetivo de construir el componente *Triples* de HDT. En este caso, es necesario reemplazar los términos RDF por su correspondiente ID en el *Diccionario* y codificar sucintamente esta representación ID-triples de acuerdo a los principios considerados en HDT. HDT-MR afronta estas necesidades mediante dos *jobs* MapReduce y dos procesos locales (ver figura 2).

Job 2.1: Construcción de ID-triples Para ejecutar este primer *job* es necesario que cada nodo disponga del Diccionario comprimido (en la etapa anterior). Una vez recibido y cargado en memoria, los *mappers* simplemente leen los términos de cada triple y los reemplazan por sus IDs (obtenidos mediante operaciones

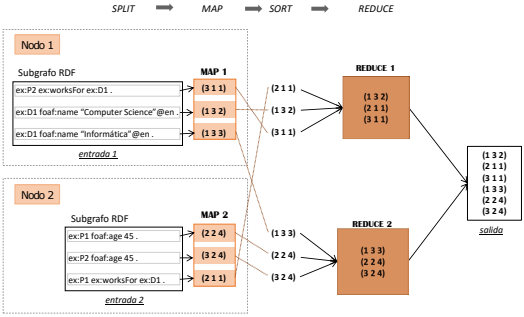


Figura 5. Job 2.1: construcción de ID-triples

Algoritmo 3 Job 2.1: construcción de ID-triples

```
function MAP(key,value)        ▷ key: line number (discarded)        ▷ value: triple
    emit(value.subject, dictionary.id(value.subject))
    emit(value.predicate, dictionary.id(value.predicate))
    emit(value.object, dictionary.id(value.object))
end function
```

de búsqueda en el Diccionario). Estos resultados se transmiten a *identity reducers* que sencillamente ordenan su entrada y emiten una lista de pares `(ID-triple, null)`, como puede verse en la figura 5 (omitimos *null* por simplicidad). El resultado final de este *job* (ver Algoritmo 3) comprende tantas listas ordenadas de ID-Triples como *reducers* hayan participado en el job.

Job 2.2: Ordenación de ID-Triples Al igual que en la primera etapa, este *job* asume la responsabilidad de ordenar la salida obtenida por el *job* previo. Antes de comenzar el *job*, se utiliza el *TotalOrderPartitioner* para muestrear la entrada y, posteriormente, ordenar los ID-triples por sujeto, preciado y objeto. Es, por tanto, un proceso sencillo que combina *identity mappers* e *identity reducers*. El resultado contiene una lista ordenada de pares (ID-triples, *null*), La figura 6 ilustra este *job* (nótese de nuevo que los valores *null* se omiten en la salida).

Proceso Local 2.3: Codificación de los Triples HDT Este proceso final se ejecuta (en el maestro) mediante un bucle que itera sobre el fichero de ID-triples, obtenido en el *job* anterior, y construye las secuencias de bits (`Bp`, `Bo`) y enteros (`Sp`, `So`), explicadas previamente.

4. Evaluación Experimental

Esta sección evalúa el rendimiento de HDT-MR y lo compara con la propuesta original (mono-nodo). Para ello, hemos desarrollado un prototipo de `HDT-MR`[7] (en

[7] Disponible en `http://goo.gl/Z5vl72`.

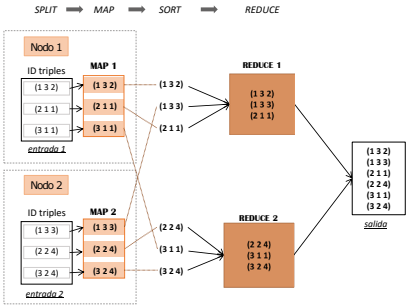


Figura 6. Job 2.2: Ordenación de ID-Triples

Máquina	Configuración
Nodo Único	Intel Xeon E5-2650v2 @ 2.60GHz (32 núcleos), 128GB RAM. Debian 7.8
Maestro	Intel Xeon X5675 @ 3.07 GHz (4 núcleos), 48GB RAM. Ubuntu 12.04.2
Esclavos	Intel Xeon X5675 @ 3.07 GHz (4 núcleos), 8GB RAM. Debian 7.7

Cuadro 1. Entorno experimental.

						Tamaño (GB)			
Colección	Triples	|SO|	|S|	|O|	|P|	NT	NT+lzo	HDT	HDT+gz
LinkedGeoData	0,27BN	41,5M	10,4M	80,3M	18,3K	38,5	4,4	6,4	1,9
DBPedia	0,43BN	22,0M	2,8M	86,9M	58,3K	61,6	8,6	6,4	2,7
Ike	0,51BN	114,5M	0	145,1K	10	100,3	4,9	4,8	0,6
Mashup	1,22BN	178,0M	13,2M	167,2M	76,6K	200,3	18,0	17,1	4,6
LUBM-1000	0,13BN	5,0M	16,7M	11,2M	18	18,0	1,3	0,7	0,2
LUBM-2000	0,27BN	10,0M	33,5M	22,3M	18	36,2	2,7	1,5	0,5
LUBM-3000	0,40BN	14,9M	50,2M	33,5M	18	54,4	4,0	2,3	0,8
LUBM-4000	0,53BN	19,9M	67,0M	44,7M	18	72,7	5,3	3,1	1,0
LUBM-5000	0,67BN	24,9M	83,7M	55,8M	18	90,9	6,6	3,9	1,3
LUBM-6000	0,80BN	29,9M	100,5M	67,0M	18	109,1	8,0	4,7	1,6
LUBM-7000	0,93BN	34,9M	117,2M	78,2M	18	127,3	9,3	5,5	1,9
LUBM-8000	1,07BN	39,8M	134,0M	89,3M	18	145,5	10,6	6,3	2,2
LUBM-12000	1,60BN	59,8M	200,9M	133,9M	18	218,8	15,9	9,6	2,9
LUBM-16000	2,14BN	79,7M	267,8M	178,6M	18	292,4	21,2	12,8	3,8
...	...	...	...	...	...	...	...	...	...
LUBM-68000	9,05BN	337,9M	1202,4M	757,0M	18	1244,4	90,2	57,6	18,9
LUBM-72000	9,59BN	357,8M	1269,4M	801,7M	18	1318,0	95,5	61,3	20,0

Cuadro 2. Descripción de los conjuntos de datos de prueba.

Hadoop, versión 1.2.1) que usa la librería HDT-Java existente[8] (RC-2). Consideramos esta misma librería HDT-Java como referencia original de la implementación mono-nodo. Nuestro entorno de experimentación (ver Cuadro 1) considera dos configuraciones computacionales. Por un lado, utilizamos un servidor potente que ejecuta la implementación HDT mono-nodo. Por otro lado, desplegamos HDT-MR sobre un cluster con *maestro* potente y 10 *esclavos* con recursos de memoria más limitados. Para una comparación más justa, la memoria disponible en el servidor (HDT mono-nodo) es la misma que en la suma de la memoria de todos de los nodos cluster Hadoop (HDT-MR).

[8] `http://code.google.com/p/hdt-java/`

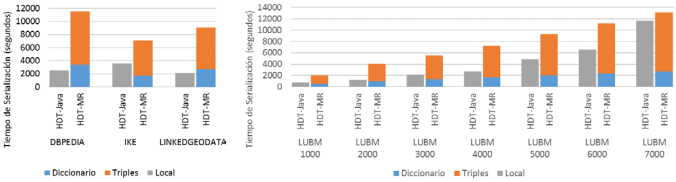


Figura 7. Tiempos de serialización: HDT-Java vs. HDT-MR.

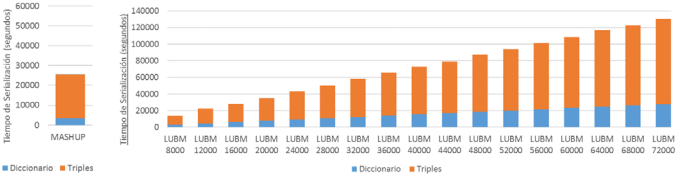


Figura 8. Tiempos de serialización: HDT-MR.

En lo referente a las colecciones de datos de prueba (ver Cuadro 2), consideramos un corpus variado, compuesto de colecciones reales y sintéticas. La elección de las colecciones reales considera criterios de volumen y variedad, así como su uso previo para evaluación en el estado del arte. *Ike*[9] incluye mediciones meteorológicas del huracán Ike; *LinkedGeoData*[10] es un gran conjunto de datos espaciales derivado de *Open Street Map*; y DBPedia 3.8[11] es una base de conocimiento extraída de Wikipedia. De igual modo, combinamos todos ellos en un *mashup* que incluye los datos de las tres fuentes anteriores. Por otra parte, empleamos la herramienta LUBM [6] como generador de datos sintéticos (basado en modelar un sistema académico con un número parametrizable de universidades). Construimos "colecciones pequeñas" desde 1000 (0, 13 billones[12] de triples) a 8000 universidades (1, 07 billones de triples) y "colecciones grandes" (añadiendo 4000 universidades en cada una de ellas: 0, 55 billones de triples), hasta un máximo de 72000 universidades (9, 59 billones de triples).

El Cuadro 2 muestra los tamaños originales en formato NTriples (NT) y los comprimidos con `lzo`. Cabe destacar que `HDT-MR` usa `lzo` para comprimir las colecciones antes de almacenarlas en HDFS. Como se muestra en el cuadro, la colección más grande ocupa 1318 GB en NTriples, mientras que comprimida con `lzo` apenas requiere 95, 5 GB.

[9] `http://wiki.knoesis.org/index.php/LinkedSensorData`
[10] `http://linkedgeodata.org/Datasets`, versión 01-07-2013
[11] `http://wiki.dbpedia.org/Downloads38`
[12] Por motivo de consistencia con el estado del arte, empleamos la escala anglosajona, esto es, 1 billón = 1 000 000 000 (mil millones).

La figura 7 compara los tiempos de serialización de HDT-Java y HDT-MR en las colecciones de menor tamaño, mientras que la Figura 8 sólo muestra los tiempos de HDT-MR, dado que HDT-Java no es capaz de serializar las colecciones más grandes. HDT-Java obtiene un buen rendimiento en los conjuntos de datos reales, y HDT-MR sólo puede competir en el conjunto *Ike*. No obstante, no se trata de un resultado inesperado, puesto que HDT-Java se ejecuta en memoria principal mientras que HDT-MR paga el sobrecoste de las operaciones de I/O que realiza Hadoop. En cambio, HDT-Java no es capaz de serializar el *mashup*, ya que los 128 GB de RAM disponibles son insuficientes para procesar tal volumen de información en un único nodo. El resultado es similar para las colecciones sintéticas LUBM: HDT-Java es la mejor opción para los conjuntos de datos pequeños, pero la diferencia respecto a HDT-MR se reduce a medida que el tamaño del conjunto de datos se incrementa. HDT-Java, además, no escala para colecciones mayores que *LUBM-8000* (1,07 billones de triples). Este es el escenario indicado para HDT-MR, que consigue procesar sin problemas todas las colecciones hasta *LUBM-72000*. Como muestran ambas figuras, los tiempos de serialización se incrementan de manera lineal con el tamaño de la *colección*, siendo la codificación de los Triples la etapa más costosa.

Aunque la compresión no es el objetivo principal de este artículo, cabe mencionar los resultados obtenidos por HDT, ya que ninguno de los experimentos previos reportaba número para colecciones tan grandes. El cuadro 2 resume dichos resultados y puede observarse que HDT mejora los espacios obtenidos por `lzo` en todos los casos. La diferencia es aún mayor cuando HDT se combina con gzip. Por ejemplo, HDT utiliza 19,7GB menos que `NT+lzo` para comprimir *LUBM-40000* y la diferencia se incrementa hasta 42,5GB frente a `HDT+gz`.

5. Conclusiones

HDT está adquiriendo un reconocimiento cada vez mayor como una referencia *de facto* en el área de compresión RDF. Además, los índices que incluye HDT permiten recuperar RDF sin necesidad de descompresión previa, por lo que han surgido aplicaciones semánticas que emplean HDT como motor de almacenamiento RDF. En este artículo presentamos HDT-MR, una técnica para resolver los problemas de escalabilidad que surgen al construir HDT a gran escala. HDT-MR reduce el consumo de memoria empleado originalmente en su construcción, trasladando esta tarea al paradigma MapReduce. En este trabajo, presentamos el *workflow* distribuido de HDT-MR, evaluamos su rendimiento frente a una solución en un único nodo, en colecciones de hasta 9000 millones de triples RDF. Los resultados muestran como HDT-MR escala a cualquier tamaño en *clusters* con hardware de uso común, mientras que la solución original en un único nodo no es capaz de procesar tamaños mayores que 1000 millones de triples. HDT-MR, por tanto, reduce los requisitos hardware para procesar Big Semantic Data.

Nuestro trabajo futuro considera explotar los resultados de HDT-MR ya que la comunidad HDT puede disponer de colecciones HDT más grandes, promoviendo con ello el desarrollo de nuevas aplicaciones a gran escala. De igual modo,

consideramos combinar HDT y MapReduce para otras tareas como la consulta y razonamiento en Big Semantic Data.

Agradecimientos

Este proyecto ha sido financiado por el programa de innovación e investigación Horizon 2020 de la Unión Europea gracias a la ayuda Marie Skłodowska-Curie No 642795, Austrian Science Fund (FWF): M1720-G11 y el Ministerio de Economía y Competitividad: TIN2013-46238-C4-3-R.

Referencias

1. N.R. Brisaboa, A. Cerdeira-Pena, A. Fariña, and G. Navarro. A Compact RDF Store using Suffix Arrays. In *String Processing and Information Retrieval*, pages 103–115, 2015.
2. L. Cheng, A. Malik, S. Kotoulas, T.E. Ward, and G. Theodoropoulos. Efficient Parallel Dictionary Encoding for RDF Data. In *Proc. of WebDB*, 2014.
3. O. Curé, G. Blin, D. Revuz, and D.C. Faye. WaterFowl: A Compact, Self-indexed and Inference-Enabled Immutable RDF Store. In *Proc. of ESWC*, pages 302–316, 2014.
4. J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Largee Clusters. In *Proc. of OSDI*, pages 137–150, 2004.
5. J.D. Fernández, M.A. Martínez-Prieto, C. Gutiérrez, A. Polleres, and M. Arias. Binary RDF Representation for Publication and Exchange. *Journal of Web Semantics*, 19:22–41, 2013.
6. Y. Guo, Z. Pan, and J. Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics*, 3(2):158–182, 2005.
7. A. Joshi, P. Hitzler, and G. Dong. Logical Linked Data Compression. In *Proc. of ESWC*, pages 170–184, 2013.
8. F. Manola and R. Miller. *RDF Primer*. W3C Recommendation, 2004.
9. M.A. Martínez-Prieto, M. Arias, and J.D. Fernández. Exchange and Consumption of Huge RDF Data. In *Proc. of ESWC*, pages 437–452, 2012.
10. M.A. Martínez-Prieto, N.R. Brisaboa, F. Claude, R. Cánovas, and G. Navarro. Practical Compressed String Dictionaries. *Information Systems*, 56:73–108, 2016.
11. J.Z. Pan, J.M. Gómez-Pérez, Y. Ren, H. Wu, and M. Zhu. SSP: Compressing RDF Data by Summarisation, Serialisation and Predictive Encoding. Technical report, 2014.
12. J. Urbani, J. Maassen, H. Bal, N. Drost, F. Seintra, and H. Bal. Scalable RDF Data Compression with MapReduce. *Concurrency and Computation: Practice and Experience*, 25:24–39, 2013.
13. R. Verborgh, O. Hartig, B. De Meester, G. Haesendonck, L. De Vocht, M. Vander Sande, R. Cyganiak, P. Colpaert, E. Mannens, and R. Van de Walle. Querying Datasets on the Web with High Availability. In *Proc. of ISWC*, pages 180–196, 2014.
14. S. Álvarez García, N. Brisaboa, J.D. Fernández, M.A. Martínez-Prieto, and G. Navarro. Compressed Vertical Partitioning for Efficient RDF Management. *Knowledge and Information Systems*, 2014.

Arquitectura Software Basada en Tecnologías Smart para Agricultura de Precisión

Miguel Sánchez[1], Manuel Barrena[1], Pablo Bustos[2], Carlos Campillo[3], Pablo García[1]

[1] Dpto. Ingeniería de Sistemas Informáticos y Telemáticos
[2] Dpto. Tecnología de Computadores y de las Comunicaciones
Escuela Politécnica, Universidad de Extremadura, C/Avda. de la Universidad, s/n. 10003 Cáceres, España
[3]Centro de Investigaciones Científicas y Tecnológicas de Extremadura (CICYTEX)

{mscabrera, barrena, pbustos, pablogr}@unex.es
carlos.campillo@gobex.es

Resumen. Este artículo describe la arquitectura de un sistema de información en el contexto de la agricultura de precisión. Una red de sensores instalados sobre las zonas de cultivo se encarga de monitorizar las variables que finalmente alimentan el modelo de riego y fertilización implementado. Las mediciones obtenidas se almacenan de manera autónoma y continua sobre una base de datos MongoDB, cuyo diseño prevé la variabilidad espacio-temporal de los diversos componentes de la aplicación (zonas, cultivos, sectores de irrigación, etc.). Datos obtenidos de otras fuentes, tales como servicios meteorológicos o análisis de suelo completan el modelo, cuyo objetivo final es el de mejorar la eficiencia en la gestión de la explotación agraria. Los procesos que combinan toda esta información y ponen en marcha el modelo se implantan mediante el uso del framework de edición de flujos Node-RED, con el desarrollo de flujos de datos para establecer la conexión con la red de sensores y servicios meteorológicos y proveer de datos al sistema, consiguiendo al integrar estas tecnologías una infraestructura digital para la explotación rentable de recursos agrarios.

Palabras Clave: Agricultura de Precisión, Sistemas de Información Agrícolas, Tecnologías Smart.

1 Introducción

El grado de robustez que en poco tiempo han alcanzado modernos paradigmas como el Internet de las Cosas, Computación en la Nube, Big Data y en general todas aquellas que suelen agruparse bajo la denominación de tecnologías Smart, ha disparado la presencia y contribución de las nuevas tecnologías de la información y las comunicaciones (TIC) por supuesto en nuestro entorno más inmediato, pero interesantemente también en dominios de aplicación menos visibles para el ciudadano común. El mundo rural y particularmente el sector agrícola es un claro ejemplo donde

la incorporación de estas tecnologías están favoreciendo su evolución hacia una gestión mucho más inteligente y sostenible [1,2,3]. De hecho el término "Agricultura de Precisión" se acuña en este contexto para definir una nueva forma de gestionar las explotaciones agrarias mediante la intervención de una serie de tecnologías de aplicación en la producción agraria, que tienen como factor común el uso de las TIC en la racionalización de la toma de decisiones y su precisa ejecución [4].

En el ámbito de la agricultura de precisión, los medios que posibilitan una gestión más tecnificada de la actividad agrícola incluyen por lo general imágenes de satélite, sensores de medida continua instalados en las parcelas, servicios de meteorología, volcado de datos a distancia, gestión de información vía web o utilización de smartphones como instrumentos de entrada y salida de datos. La combinación de estos elementos proporciona a técnicos y agricultores una oportunidad de mejorar y administrar de forma eficiente una gran cantidad de información disponible, ofreciendo por tanto la oportunidad de optimizar la gestión de estas zonas agrícolas desde un punto de vista agronómico (que afecte de forma directa y beneficiosa a los cultivos), medioambiental (reduciendo el impacto relacionado con las actividades agrícolas) y económico (mejorando el rendimiento de la producción).

Para mejorar la gestión de las explotaciones, la agricultura de precisión atiende por lo general a dos importantes aspectos problemáticos [1,4]. Por una parte la heterogeneidad espacial de las parcelas agrícolas, la cual genera diferencias de desarrollo y producción de los cultivos, a pesar de haber realizado homogéneamente las operaciones de cultivo (laboreo, siembra, riego, abonado, etc.), por otra parte la imprecisión en la realización de las actividades por parte del agricultor, basadas en apreciaciones personales sustentadas por una información habitualmente difusa del estado del cultivo, lo cual provoca con frecuencia un uso desproporcionado de los recursos (agua, fertilizantes, etc.). El tratamiento de la heterogeneidad espacial ha permitido saltar desde las parcelas de ensayo a explotaciones reales [5], donde las circunstancias resultan a priori mucho menos controlables, mientras que la capacidad de disponer de una información precisa sobre el estado de la explotación y los cultivos, permite guiar al agricultor en sus actividades, favoreciendo su planificación y fomentando el equilibrio en el aprovechamiento de recursos y el uso eficiente de los mismos. En la agricultura, el agua es el recurso más importante y su gestión de forma eficiente es uno de los objetivos prioritarios de este sector. Aplicar un sistema eficiente de riego no es tarea sencilla, ya que tanto el riego en exceso como aplicar menos agua de la debida conlleva resultados perjudiciales para el cultivo y también para el rendimiento de la explotación. La agricultura de precisión requiere pues aplicar modelos agronómicos capaces de abordar el problema del riego y las necesidades nutricionales y de crecimiento propias del cultivo que permitan desarrollar un plan de actividades óptimo.

En este artículo se presenta la arquitectura software de un sistema de información que proporciona soporte para el desarrollo de una agricultura de precisión, abordando específicamente los dos aspectos comentados anteriormente, de una parte dando respuesta a la heterogeneidad espacial y proporcionando por otra un modelo de riego y nutrición capaz realizar recomendaciones personalizadas a técnicos, lo cual redunda en una mejora de la gestión de la explotación. El sistema de información reproduce un esquema estándar [2] de monitorización de variables que miden el estado de la parcela. Este proceso de monitorización se lleva a cabo de forma autónoma, mediante

el despliegue de diversos tipos de sondas, estaciones climatológicas y servicios meteorológicos disponibles a los cuales se accede automáticamente. Los valores de estas variables, así como toda la información inherente a la parcela que se monitoriza, se hace persistente en la base de datos que sustenta al sistema de información. En consonancia con recientes aplicaciones de agricultura de precisión [3,6], un sistema de base de datos NoSQL fácilmente escalable y versátil para almacenar datos de estructura heterogénea, se encarga de registrar toda la información que genera la monitorización. Finalmente, un componente central del sistema implementa y ejecuta el modelo que obtiene como salida una planificación de las actividades de riego y fertilización para el agricultor.

Para describir la arquitectura software del sistema, el presente artículo se organiza en tres secciones. La primera de ellas lleva a cabo una descripción de los componentes principales del sistema, poniendo especial énfasis en la estructura de datos capaz de ofrecer soporte a la característica de espacio-temporalidad que presentan estas aplicaciones. Una vez descrita la arquitectura, la siguiente sección aporta algunos detalles sobre el modelo de riego que implementa el sistema y finalmente la última sección presenta como resultado experimental la aplicación de esta arquitectura en un proyecto concreto de aplicación de tecnologías integradas para la explotación rentable de recursos agrarios que ha finalizado recientemente.

2 Descripción del sistema

La arquitectura software del sistema integra el uso de varias tecnologías sobre las que se asienta la estructura general del sistema, tal y como muestra la figura 1:

- Alimentadores: Elementos provenientes de fuentes externas que nutren de información al sistema, tales como sensores y servicios de meteorología.
- Framework de edición de flujos encargado de realizar la conexión autónoma de los alimentadores al sistema, recibir la información que proporcionan, procesarla y almacenarla en la base de datos.
- Base de datos NoSQL encargada de almacenar los datos del sistema.
- Modelo de riego implementado en el sistema y encargado de analizar los diferentes datos extraídos de las fuentes externas y realizar una estimación de horas de riego sobre determinado cultivo.

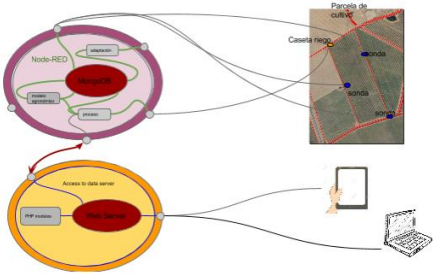


Fig. 1. Esquema de la arquitectura de la plataforma.

2.1 Alimentadores del sistema

Denominamos alimentadores (feeders) a las diferentes fuentes externas que alimentan de información al sistema de manera autónoma y constante. Podemos hablar de dos tipos diferentes de alimentadores de información, la suministrada por tecnologías de sensores y registradores de datos por un lado y por otro lado la ofrecida por los servicios de meteorología. Por un lado, los dataloggers o registradores de datos son colocados en la parcela de cultivo, resultado de un proceso previo de determinación de la zona representativa a partir de información de suelo, cubierta vegetal, mapas de desarrollo de cultivo, etc. Dependiendo del modelo concreto, cada concentrador permite la instalación de varios sensores que, según las necesidades, son elegidos para que suministren los datos necesarios al sistema. La información que envía el datalogger se corresponde con los sensores instalados. Estos sensores que emiten valores de humedad volumétrica del suelo y humedad relativa a diferente profundidad (30cm y 80cm), sensores que miden humedad relativa y temperatura situada en la parte superior de la planta y sensores de humectación de la hoja. Todos estos datos son enviados a la plataforma del datalogger a través de una comunicación GSM mediante un módem interno, como puede observarse en la figura 2.

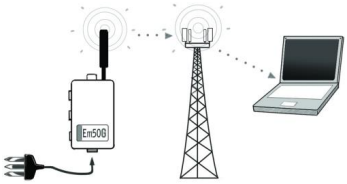


Fig. 2. Esquema de comunicación del datalogger.

Por otro lado, la información relevante para el sistema que provee de datos meteorológicos que afectan a la parcela de cultivo, tales como temperatura y humedad (máxima, mínima y media), velocidad del viento, radiación solar, pluviometría, niveles de evapotranspiración (ET-PM, ET-Rad, ET-BC y ET-H) es proporcionada por las estaciones agrometeorológicas. Si la propia explotación no cuenta con una estación de estas características, el sistema posibilita el acceso a servicios meteorológicos online, tal como las estaciones provenientes de la Red de Asesoramiento al Regante de Extremadura (Redarex) [7], desde el cual pueden realizarse consultas externas de las variables meteorológicas para las estaciones más cercanas a la parcela que se monitoriza.

Ambas fuentes externas registran los datos de manera autónoma y constante, suministrando al sistema de información heterogénea que permite trabajar y mantener un control tanto a nivel propio del cultivo (estado hídrico del suelo y de la planta) mediante los loggers, como a un nivel más externo con las variables meteorológicas.

2.2 Framework de eventos de flujos

Los procesos que se encargan de (1) la extracción de los datos proporcionados por los alimentadores, (2) la transformación de los mismos y (3) su posterior carga en la base de datos son realizados mediante el uso del framework de visualización open-source y creación de flujos Node-RED [8], desarrollada por IBM Emerging Technology, que nos permite realizar una interconexión de los diferentes componentes (dispositivos hardware, APIs, servicios online, etc.) mediante el diseño de flujos de eventos. Está construido en node.js, lo que le da la posibilidad de funcionar justo al borde de la red o en la nube, aportando flexibilidad. Mediante el ecosistema del gestor de paquetes de node (npm) es posible extender de manera sencilla las opciones de nodos disponibles para adaptar según las necesidades de los eventos a desarrollar y las operaciones que quieran realizarse.

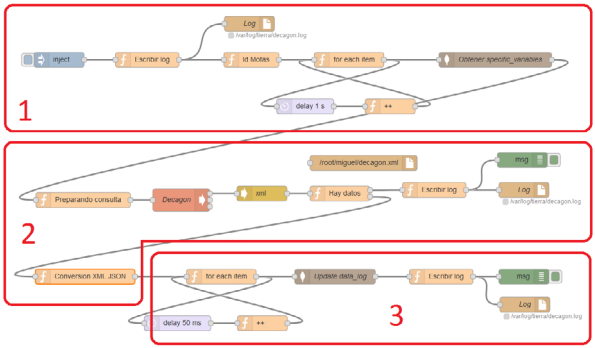


Fig. 3. Estructura del evento de flujo.

En nuestro sistema se han diseñado dos flujos de eventos para tratar de forma precisa las dos fuentes de información que suministran los alimentadores. Por un lado, aquellos datos pertenecientes a las fuentes de sensores instalados en las parcelas de cultivo, como indica la figura 3, y por otro lado, los datos de variables meteorológicas medidos por las estaciones. El diseño del flujo para el primer caso realiza el proceso de conexión a la plataforma a la que se envían los datos de los sensores mediante el uso de credenciales de acceso (disponibles al dar de alta los dataloggers en el sistema tras su instalación en las parcelas de cultivo), de este modo es posible extraer los datos que son enviados de forma continuada. El proceso de transformación de estos datos es una tarea complicada, ya que la información de las variables que miden los sensores viene dada a modo de pulsos, siendo necesario aplicar, en cada caso, fórmulas de conversión a cada uno de los sensores del datalogger para conseguir que dichos datos sean adecuados a la información con la que trabaja el sistema (valores de humedad, temperatura, etc.). Tanto este evento como el relacionado con los datos meteorológicos cargan de forma automática la información a la base de datos consiguiendo que el sistema sea alimentado ininterrumpidamente, lanzando peticiones cada 30 minutos a la plataforma de los sensores para recibir los datos y una vez al final del día para las estaciones de meteorología.

Con el desarrollo de estos eventos de flujo se consigue que el sistema contenga los datos de ambas fuentes de forma autónoma, asegurando la persistencia y disponibilidad de los mismos mediante su replicación en la base de datos.

2.3 Base de datos

Con el fin de conseguir un óptimo almacenamiento y la persistencia de los datos del sistema hemos considerado la utilización de sistemas de bases de datos NoSQL de última generación, ya que se adaptan mucho mejor que los sistemas relacionales tradicionales a los requisitos de eficiencia en aplicaciones intensivas de datos, como es nuestro caso [10].

Los sistemas de bases de datos NoSQL escalan de forma horizontal, mejorando el rendimiento del sistema al incorporar nuevos nodos al sistema, mientras que el escalado vertical añade más recursos de memoria, CPU y disco a máquinas aisladas para mejorar su capacidad de procesado, lo cual tiene un impacto mayor en el diseño del sistema.

La facilidad de diseño del modelo que nos proporciona MongoDB para el almacenamiento de los datos nos hizo decantar por este sistema como motor de almacenamiento y acceso. MongoDB es una base de datos NoSQL orientada a documentos donde el elemento base para el almacenamiento es un documento en formato JSON. El uso del formato JSON simplifica el modelo de datos, ya que cualquier dato que llega al sistema puede adecuarse fácilmente a este estándar. De este modo, cuando un dato es generado en un alimentador resulta muy sencillo de encapsular en origen en un documento JSON y enviarlo directamente a MongoDB, una vez que ha sido aplicado el proceso de transformación del mismo si fuera necesario. Pese a que una de las ventajas más importantes con respecto a las bases de datos relacionales es que no es necesario seguir un esquema, hemos establecido un modelo estándar que permite representar con fidelidad los distintos tipos de datos que

alimentan al sistema y atender las necesidades posteriores de información de los usuarios.

Además, los sistemas de bases de datos NoSQL actuales contemplan cada vez más las necesidades de tratamiento espacio-temporal incorporando en ellos mecanismos que facilitan su combinación, aunque bien es cierto que aún nos queda recorrido para llegar a solucionar por completo los problemas y dificultades que presenta este tratamiento conjunto. Sin embargo el tratamiento del tiempo deberá ser contemplado desde una lógica exterior al propio sistema, apoyada en las decisiones de diseño que hemos tomado para dotar de memoria temporal a nuestra plataforma.

De este modo hemos definido las estructuras que especifican con precisión el significado y componentes de cada uno de los objetos con los que vamos a trabajar. La utilización de esquemas posibilita describir y validar la información que se almacena en cada objeto.

El esquema genérico para los principales tipos de datos se dispone en dos bloques básicos:

- Bloque descriptivo: Contiene datos que son invariantes en el tiempo o su variación no es de interés para el sistema. De estos datos sólo se almacena su versión más reciente.
- Bloque productivo: Contiene los datos generados por algún actuador del sistema y que perviven y son válidos durante un periodo acotado de tiempo. El sistema mantiene un registro con las diferentes versiones de estos datos.

La estructura de estos esquemas puede estar formada por un tipo simple (tal como un número entero o una cadena de caracteres) o un tipo estructurado (compuesto de otros subtipos). Estos últimos a su vez pueden ser listas, bien de elementos simples o de objetos del mismo tipo y colecciones que pueden incluir objetos de diferentes tipos.

La información que almacena el sistema está organizada en cuatro tipos, cada uno de ellos con un esquema diseñado.

- Esquema para almacenar la información de las parcelas y las zonas de cultivo.
- Esquema para los agentes, usuarios con información de tipo cooperativa, técnico o propietario.
- Esquema para los feeders que alimentan al sistema y que pueden ser de tipo mota (registradores de datos) o servicio meteorológico.
- Esquema para las actividades que pueden realizarse en las zonas de cultivo, ya sea riego, fertilización, análisis de suelo, fitosanitario, etc.

2.4 Coherencia espacio-temporal

Espacio y tiempo juegan un papel muy importante y son dimensiones que hay que contemplar en el sistema. ¿Qué ocurre cuando uno o más elementos del sistema pueden verse expuestos a cambios en el tiempo? ¿Qué sucede con aquella información que ha de seguir disponible aun cuando algún elemento del sistema no lo está? Toda la información que se genera en el sistema es importante para su uso inmediato pero también, del mismo modo, para su posible uso futuro. Imaginemos los casos en los que una cooperativa agrícola necesita los datos de campañas y temporadas pasadas para generar informes estadísticos de producción, o técnicos que quieren mantener un histórico de los cultivos de una parcela que ha ido cambiando

sus zonas de cultivo con el tiempo. Todo esto es información que hay que mantener. En el ejemplo de la figura 4 se observa una parcela que inicialmente tiene cuatro zonas de cultivo y que evoluciona en el tiempo transformando su superficie en diferentes zonas.

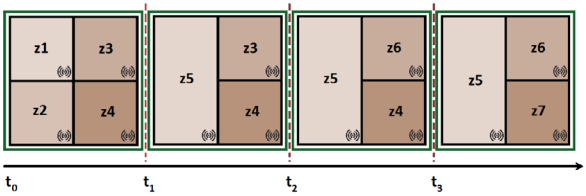


Fig. 4. Cambios en las zonas de cultivo de una parcela a lo lardo del tiempo.

Inicialmente la parcela consta de cuatro zonas (z1, z2, z3 y z4) pero con el paso del tiempo, ya sea por la llegada de nuevas temporadas de cultivo o por algún otro motivo, las zonas van cambiando hasta llegar a una distribución final de tres zonas (z5, z6 y z7). La necesidad de reflejar cada uno de los cambios sufridos por los elementos del sistema durante el tiempo es clave para no perder información y mantener una disponibilidad absoluta del estado de los datos en todo momento.

Por ello, definir las estructuras que van a dar cabida a los datos siguiendo un criterio de coherencia espacio-tiempo es necesario para que nuestro sistema sea capaz de responder a consultas sobre lo sucedido en el pasado sobre aquellos elementos que estén sujetos a cambios y cuya información haya que mantener de forma continua en el sistema. Para conseguir esta conexión espacio-tiempo, los elementos principales como las parcelas de cultivo sobre las que se aplican modelos agronómicos y los registradores de datos instalados en ellas, tienen bien definidos sobre el esquema de sus estructuras la variante espacial que permite almacenar información acerca de su localización geográfica y delimitar la extensión del terreno.

En el aspecto temporal, la información que se almacena en nuestra base de datos lleva definida en su esquema una marca que indica la validez de dicha información en el tiempo.

De este modo, en el ejemplo de la figura 5 apartado a), se observa la forma en la cual el sistema afronta los cambios en el tiempo sobre una parcela y su división en zonas de cultivo. El sistema almacena un histórico de los cambios en las zonas de cultivo de una parcela combinando la validez temporal de cada una de ellas con una marca que acota y limita su existencia en el sistema. De forma inicial, cada zona de cultivo lleva adherida una fecha de inicio que refleja el instante en el que dicha zona fue creada. Del mismo modo, cada zona tiene una fecha de fin de existencia que inicialmente no está activa si la zona no sufre cambios en el tiempo, y que pasará a activarse cuando alguna causa externa provoque que la zona deje de estar disponible. Esto le indica al sistema qué evolución ha sufrido una parcela y en qué instantes, conociendo en todo momento un histórico de los datos y el estado en el que se encuentra.

Fig. 5. a) Histórico de las localizaciones geográficas de un datalogger en el tiempo. b) Histórico de las localizaciones geográficas de un datalogger en el tiempo.

Otro elemento del sistema que puede sufrir cambios en el tiempo son los dataloggers instalados en las parcelas de cultivo. ¿Qué ocurre si un registrador de datos cambia de localización geográfica y es instalado en otra zona de cultivo dentro de la parcela? Como en el caso anterior, se impone la necesidad de mantener en el sistema todos los datos que registran los sensores. De este modo, la estructura del elemento cambiante es capaz de afrontar una evolución temporal sin que la información del sistema sufra modificaciones. Como se observa en la figura 5, apartado b), el registrador de datos en el instante t_1 no tiene activada la marca de tiempo que acota su actividad, lo cual indica que, para esa localización geográfica donde está instalado, el elemento sigue emitiendo información. En el instante t_2 podemos ver cómo el registrador de datos cambia de localización geográfica indicando al sistema la actividad del mismo. Se activa la marca de tiempo para indicar al sistema qué localización ha dejado de ser válida y a partir de qué nueva localización se está recibiendo información. El histórico del elemento registrador de datos mantiene los cambios sufridos en el tiempo y el sistema deja disponible la información registrada por el elemento en localizaciones pasadas. De este modo, pese a que algunos de los elementos que forman el sistema puedan estar sujetos a sufrir cambios en el tiempo, la coherencia del criterio espacio-temporal se mantiene, ya que toda la evolución del elemento queda registrada, indicando los periodos de validez de cada uno, y conservando los datos siempre disponibles en el sistema.

3 Modelo de riego

El modelo agronómico de riego utiliza la información disponible en el sistema proveniente de las fuentes externas que proporcionan los alimentadores (red de sensores y estaciones de meteorología) e información específica de la parcela y el cultivo, estableciendo diferentes niveles de datos para realizar un exhaustivo análisis del cultivo de una parcela ofreciendo como resultado de dicho proceso una estimación diaria de horas de riego recomendadas según las características y el estado del cultivo.

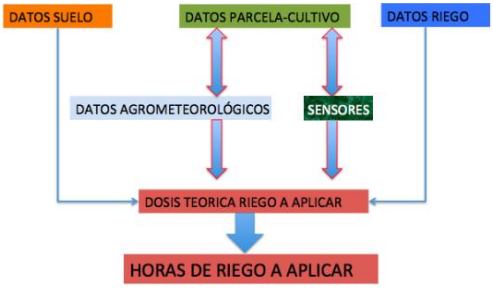


Fig. 6. Diagrama del modelo de riego.

El modelo de riego utilizado está validado para el tipo de cultivo al que se aplica, en este caso, frutales con hueso, tales como ciruelos, donde se analizan los resultados del sistema para ajustar sus posibles derivas. Por otra parte, el propio modelo se ajusta en función del tipo de riego aplicado (inundación, goteo, aspersión, etc.).

La estructuración de niveles de datos en el modelo puede establecerse del siguiente modo, acorde con la figura 6:

- Esquema para las actividades que pueden realizarse en las zonas de cultivo, ya sea riego, fertilización, análisis de suelo, fitosanitario, etc.
- Nivel 1: Se corresponde con la entrada de datos en el sistema. Información propia y necesaria de cada parcela y cultivo, tal como el tipo de cultivo, variedad, fecha de las fases (brotación, desarrollo foliar, recolección, caída de la hora, etc.).
- Nivel 2: Datos pertenecientes a la red de sensores y estaciones de meteorología. En el caso de la información perteneciente a las estaciones, el sistema almacena un histórico de los datos de las estaciones desde alrededor de una década, pudiendo aplicar un valor medio de los mismos en el caso de faltar algún dato en un momento dado.

Los datos del nivel 1 y nivel 2 permitirán obtener los valores teóricos de riego para la parcela y momento del cultivo, estos valores se convertirán a través de algoritmos de cálculo en horas de riego diarias o semanales a aplicar en el cultivo.

- Nivel 3: Sensores automáticos o manuales de estimación del estado hídrico del cultivo que pueden ajustar los algoritmos de cálculos de los valores de dosis teóricas a horas de riego.

El presente modelo ha sido desarrollado en PHP v5.6.20 y diseñado para ser lanzado en el sistema como un proceso diario que analice la información y proporcione como resultado la recomendación diaria de horas de riego para un cultivo concreto dentro de una determinada parcela.

Fecha	Día	ET0	Kc	T. R. Teórico	R. m3 Teórico	T. R. Aplicado	R. m3 Aplicado	Balance
2015-05-03	64	3.030	0.616	00:00:00	0.000	00:00:00	0.000	-46.77
2015-05-04	65	4.030	0.627	00:00:00	0.000	00:00:00	0.000	-49.58
2015-05-05	66	3.850	0.637	00:00:00	0.000	00:00:00	0.000	-52.31
2015-05-06	67	4.610	0.648	01:30:00	0.480	00:00:00	0.000	-55.63
2015-05-07	68	6.070	0.658	05:00:00	1.600	00:00:00	0.000	-60.07
2015-05-08	69	4.030	0.669	06:00:00	1.920	00:00:00	0.000	-63.07

Fig. 7. Modelo de riego. Tiempo de riego teórico.

La imagen de la figura 7 muestra la información correspondiente tras aplicar el modelo de riego a un cultivo, organizado a modo de tabla. Cada fila se corresponde con una ejecución del modelo. De este modo, se pueden observar diferentes columnas donde se indica la fecha de ejecución del mismo, día de las fases del cultivo y variables propias del modelo de riego como el coeficiente de cultivo (Kc), evapotranspiración potencial diaria (ET0), balance hídrico, etc. Las columnas con borde color rojo nos indican en horas, tanto el tiempo de riego teórico como el tiempo de riego aplicado. En este caso, se observa cómo el modelo recomienda una serie de horas teóricas y la columna de tiempo aplicado no muestra actividad de riego, indicando que el cultivo no ha sido regado esos días.

Las recomendaciones diarias que propone el modelo facilitan la labor de riego del técnico o agricultor ya que están basadas en un exhaustivo estudio y análisis de importantes variables ligadas al crecimiento y avance del cultivo, el estado del suelo y la actividad climática de la zona en la que está situada la parcela.

4 Resultados

La arquitectura software descrita en este artículo ha sido implantada en un proyecto de desarrollo que integra tecnologías con el fin de lograr una explotación rentable de los recursos agrarios (TIERRA). El proyecto contempla en su primera fase la utilización de parcelas con diferentes tipos de cultivo en Extremadura, concretamente

sobre parcelas situadas en municipios de la provincia de Badajoz (Don Benito, Villanueva de la Serena y Lobón).

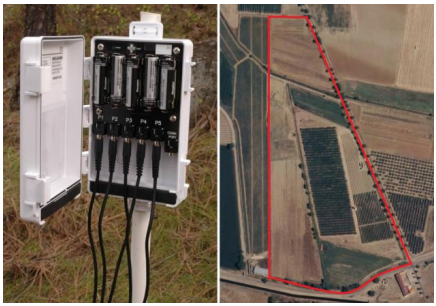


Fig. 8. Registrador de datos y parcela de cultivo.

En ellas, han sido instalados una serie de dataloggers de la marca Decagon (modelo Em50g), tal y como muestra la figura 8 en la imagen de la izquierda, que registran información del terreno y el cultivo del ciruelo en sus variedades Crimson Glo y Angeleno. Junto con esta tecnología de sensores y la información proveniente de la red de asesoramiento al regante, REDAREX, mediante sus estaciones meteorológicas, se alimenta al sistema de forma autónoma con los datos necesarios. Esto hace posible implantar la arquitectura de software basada en las tecnologías descritas, dotando a la infraestructura de un potencial aplicable a un caso real en un proyecto agronómico para agricultura de precisión. El proyecto TIERRA cuenta además con el desarrollo de la aplicación Itierra para ofrecer las funcionalidades del sistema al usuario. La interfaz visual, disponible tanto para la versión web como para dispositivos smartphone y tablet, facilita por supuesto las tareas de alta y registro de usuarios y alimentadores que se encargarán de monitorizar las variables de interés para el sistema. Mediante el uso de la API de Google Maps y SIGPAC [9], se simplifica sobremanera el registro de las parcelas y su georreferenciación y un formulario completo y extenso le facilita al usuario la introducción de datos relativos al cultivo, características de suelo, etc. La aplicación permite de modo sencillo e intuitivo la creación de actividades programadas entre las que se contempla por supuesto la planificación del riego que propone de modo automático el modelo. La figura 9 presenta una captura de pantalla correspondiente a esta sección de la aplicación.

Aunque el alcance del proyecto referido contempla en exclusiva el despliegue del sistema en parcelas determinadas, la implantación del mismo prevé su extensión al conjunto de explotaciones gestionadas por las cooperativas que han participado en el proyecto, así como la incorporación de nuevas funcionalidades propias de la gestión de este tipo de explotaciones. La arquitectura descrita se ha diseñado para este propósito, permitiendo la ejecución eficiente de consultas de muy variada tipología.

En lo que respecta a los resultados obtenidos, costes del sistema de información agrícola y rentabilidad al agricultor, hay que señalar que actualmente el proyecto se

encuentra en fase de pruebas, por lo que, lamentablemente, no es posible analizar todavía y con los resultados aplicados a una campaña de este tipo, la información necesaria para poder aplicar aspectos económicos y generar informes que evalúen los costes y ahorros.

Fig. 10. Aplicación Itierra. Menú de actividades programadas.

5 Conclusiones

El grado de desarrollo que están alcanzando las tecnologías denominadas Smart ha provocado un nuevo impulso en áreas de aplicación que típicamente han mostrado un grado de resistencia a la innovación. En concreto el sector agroindustrial se ha visto favorecido por este hecho y un nuevo paradigma como el de la agricultura de precisión comienza a imponerse en el sector. Los sistemas de información orientados al sector agrícola facilitan la gestión de las explotaciones, introduciendo nuevos elementos que facilitan el proceso de toma de decisiones al objeto de hacerlas más rentables y sostenibles.

En este artículo hemos presentado la arquitectura software de un sistema de estas características basado en tecnologías Smart. Como puede observarse, la implementación del sistema no requiere un despliegue desmesurado de tecnologías y su implantación no resulta compleja ni costosa en términos económicos. Aparte del modelo de riego y fertilización, la componente más delicada de diseñar desde el punto de vista del software es el esquema de información que sostiene la base de datos, ya que el mantenimiento de la coherencia espacio-temporal requiere un modelo de datos capaz de contemplar con suficiente versatilidad las dimensiones espacial y temporal que impone la aplicación. El apartado de resultados aporta finalmente algunos detalles sobre la implantación de nuestro sistema en explotaciones reales pertenecientes a cooperativas pacenses.

Agradecimientos: Este trabajo ha sido co-financiado con fondos FEDER de la Unión Europea (Proyecto Coinvestiga ref EI-14-0007-1), fondos de la Junta de Extremadura (Secretaría Gral de Ciencia, Tecnología e Innovación) y las empresas CREX, Fruvegal y Mastercom.

6 Referencias

1 Mondal, P.; Basu, M.; Bhadoria, P. B. S. Critical Review of Precision Agriculture Technologies and Its Scope of Adoption in India. *American Journal of Experimental Agriculture*, Vol. 1, No. 3, pp. 49-68 (2011)

2 Fountas, S.; Sorensen, C. G.; Tsiropoulos, Z.; Cavalaris, C.; Liakos, V.; Gemtos, T. Farm machinery management information system. *Computers and Electronics in Agriculture*, Vol. 110, pp. 131–138 (2015)

3 Kaloxylos, A.; Groumas, A.; Sarris, V.; Katsikas, L.; Magdalinos, P.; Antoniou, E.; Politopoulou, Z.; Wolfert, S.; Brewster, C.; Eigenmann, R.; Maestre Terol, C. A cloud-based Farm Management System: Architecture and implementation. *Computers and Electronics in Agriculture*, Vol. 100, pp. 168–179 (2014)

4 Juan Agüera Vega; Manuel Pérez Ruiz. Agricultura de precisión: hacia la integración de datos espaciales en la producción agraria. *Ambienta*, Vol. 105, pp. 16–27 (2013)

5 Kaloxylos, A.; Eigenmann, R.; Teye, F.; Politopoulou, Z.; Wolfert, S.; Shrank, C.; Dillinger, M.; Lampropoulou, I.; Antoniou, E.; Personen, L.; Nicole, H.; Thomas, F.; Alonistioti, N.; Kormentzas, G. Farm management systems and the Future Internet era. *Computers and Electronics in Agriculture*, Vol. 89, pp. 130–144 (2012)

6 Erena, M.; Lopez-Francos, A.; Montesinos, S.; Berthoumieu, J.: The use of remote sensing imagery and geographic information systems for irrigation management in southern Europe. *Options Méditerranéennes*, Vol. Serie B, No. 67, pp. 121-231 (2012)

7 Red de Asesoramiento al Regante de Extremadura. Redarex Plus Web. http://redarexplus.gobex.es/RedarexPlus/. Accedido el 21 de Abril de 2016

8 IBM Emerging Technologies. *Node-RED Web*. http://www.nodered.org. Accedido el 19 de Abril de 2016

9 Sistema de Información Geográfica de Parcelas Agrícolas. http://www.magrama.gob.es/es/agricultura/temas/sistema-de-informacion-geografica-de-parcelas-agricolas-sigpac-/. Accedido el 27 de Abril de 2016

10 Sharma V.; Dave M. SQL and NoSQL Databases. *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 2, Issue 8 (2012)

Distance Range Queries in SpatialHadoop

Francisco García-García[1,⋆], Antonio Corral[1,⋆], Luis Iribarne[1,⋆], and Michael Vassilakopoulos[2,⋆]

[1] Dept. of Informatics, University of Almeria, Almeria, Spain.
E-mail: {paco.garcia,acorral,liribarn}@ual.es
[2] Dept. of Electrical and Computer Engineering, University of Thessaly, Volos, Greece. E-mail: mvasilako@uth.gr

Abstract. Efficient processing of Distance Range Queries ($DRQs$) is of great importance in spatial databases due to the wide area of applications. This type of spatial query is characterized by a *distance range* over one or two datasets. The most representative and known $DRQs$ are the ε Distance Range Query (εDRQ) and the ε Distance Range Join Query ($\varepsilon DRJQ$). Given the increasing volume of spatial data, it is difficult to perform a DRQ on a centralized machine efficiently. Moreover, the $\varepsilon DRJQ$ is an expensive spatial operation, since it can be considered a combination of the εDR and the spatial join queries. For this reason, this paper addresses the problem of computing $DRQs$ on big spatial datasets in SpatialHadoop, an extension of Hadoop that supports spatial operations efficiently, and proposes new algorithms in SpatialHadoop to perform efficient parallel $DRQs$ on large-scale spatial datasets. We have evaluated the performance of the proposed algorithms in several situations with big synthetic and real-world datasets. The experiments have demonstrated the efficiency and scalability of our proposal.

Keywords: Distance Range Queries, Spatial Data Processing, SpatialHadoop, MapReduce

1 Introduction

Distance Range Queries ($DRQs$) have received considerable attention from the database community, due to its importance in numerous applications, such as spatial databases and GIS [1]. For the ε Distance Range Query (εDRQ), given one point dataset P, one query point q and a distance range $[\varepsilon_1, \varepsilon_2]$, this spatial query finds all points of P that are contained inside a region of circular shape or *annulus* centered in q with radios ε_1 and ε_2 (i.e. the region in the plane between two concentric circles of different radius). For the case of the ε Distance Range Join Query ($\varepsilon DRJQ$), given two point datasets P and Q and a distance range $[\varepsilon_1, \varepsilon_2]$, this distance-based join query finds all the possible pairs of points from $P \times Q$, having a distance between ε_1 and ε_2 of each other. Lots of researches

⋆ Work funded by the MINECO research project [TIN2013-41576-R] and the Junta de Andalucia research project [P10-TIC-6114]

have worked on the improvement of the performance of $DRQs$ by proposing efficient algorithms or designing new complex spatial index structures. However, all these approaches focus on methods that are to be executed in a centralized environment.

With the fast increase in the scale of big input datasets, processing large data in parallel and distributed fashions is becoming a popular practice. A number of parallel algorithms for Spatial Join [2, 3], K Nearest Neighbor (KNN) Join [4, 5], K Closest Pair Query (KCPQ) [6], top-K Similarity Join [7] and Similarity Join [8] in MapReduce [9] have been designed and implemented recently. But, to the authors' knowledge, there is no research works on parallel and distributed $DRQs$ (εDRQ and εDJQ) in large spatial data, which is a challenging task and becoming increasingly essential as datasets continue growing.

Actually, parallel and distributed computing using shared-nothing clusters on extreme-scale data is becoming a dominating trend in the context of data processing and analysis. MapReduce [9] is a framework for processing and managing large-scale datasets in a distributed cluster, which has been used for applications such as generating search indexes, document clustering, access log analysis, and various other forms of data analysis [10]. MapReduce was introduced with the goal of providing a simple yet powerful parallel and distributed computing paradigm, providing good scalability and fault tolerance mechanisms. The success of MapReduce stems from hiding the details of parallelization, fault tolerance, and load balancing in a simple programming framework.

However, as also indicated in [11], MapReduce has weaknesses related to efficiency when it needs to be applied to spatial data. A main shortcoming is the lack of any indexing mechanism that would allow selective access to specific regions of spatial data, which would in turn yield more efficient query processing algorithms. A recent solution to this problem is an extension of Hadoop, called SpatialHadoop [12], which is a framework that inherently supports spatial indexing on top of Hadoop. In SpatialHadoop, spatial data is deliberately partitioned and distributed to nodes, so that data with spatial proximity is placed in the same partition. Moreover, the generated partitions are indexed, thereby enabling the design of efficient query processing algorithms that access only part of the data and still return the correct result query. As demonstrated in [12], various algorithms are proposed for spatial queries, such as range, KNN, spatial joins, skyline[13] and KCP [6] queries. Efficient processing of $DRQs$ (εDRQ and εDJQ) over large-scale spatial datasets is a challenging task, and it is the main target of this paper.

Motivated by these observations, we first propose new parallel algorithms, based on plane-sweep technique, for the εDRQ and $\varepsilon DRJQ$ in SpatialHadoop on big spatial datasets. And next, we present the execution of a set of experiments that demonstrate the efficiency and scalability of our proposal using big synthetic and real-world points datasets.

This paper is organized as follows. In Section 2 we review related work on Hadoop systems that support spatial operations, the specific spatial queries using MapReduce and provide the motivation for this paper. In Section 3, we

present preliminary concepts related to $DRQs$ and SpatialHadoop. In section 4 the parallel algorithms for processing εDRQ and $\varepsilon DRJQ$ in SpatialHadoop are proposed. In Section 5, we present representative results of the extensive experimentation that we have performed, using real-world and synthetic datasets, for comparing the efficiency of the proposed algorithms, taking into account different performance parameters. Finally, in Section 6 we provide the conclusions arising from our work and discuss related future work directions.

2 Related Work and Motivation

Researchers, developers and practitioners worldwide have started to take advantage of the MapReduce environment in supporting large-scale data processing. The most important contributions in the context of scalable spatial data processing are the following prototypes: (1) *Parallel-Secondo* [14] is a parallel spatial DBMS that uses Hadoop as a distributed task scheduler; (2) *Hadoop-GIS* [15] extends Hive [16], a data warehouse infrastructure built on top of Hadoop with a uniform grid index for range queries, spatial joins and other spatial operations. It adopts Hadoop Streaming framework and integrates several open source software packages for spatial indexing and geometry computation; (3) *SpatialHadoop* [12] is a full-fledged MapReduce framework with native support for spatial data. It tightly integrates well-known spatial operations (including indexing and joins) into Hadoop; (4) *SpatialSpark* [17] is a lightweight implementation of several spatial operations on top of the *Apache Spark*[1] in-memory big data system. It targets at in-memory processing for higher performance; and (5) *GeoSpark* [18] is an in-memory cluster computing system for processing large-scale spatial data, and it extends the core of *Apache Spark* to support spatial data types, indexes, and operations. It is important to highlight that these five systems differ significantly in terms of distributed computing platforms, data access models, programming languages and the underlying computational geometry libraries.

Actually, there are several works on specific spatial queries using MapReduce. This programming framework adopts a flexible computation model with a simple interface consisting of *map* and *reduce* functions whose implementations can be customized by application developers. Therefore, the main idea is to develop *map* and *reduce* functions for the required spatial operation, which will be executed on-top of an existing Hadoop cluster. Examples of such works on specific spatial queries include: (1) *Region query* [19, 20], where the input file is scanned, and each record is compared against the query range. (2) *KNN query* [20, 21], where a brute force approach calculates the distance to each point and selects the nearest K points [20], while another approach partitions points using a Voronoi diagram and finds the answer in partitions close to the query point [21]. (3) *Skyline query* [13, 24, 25]; in [24] the authors propose algorithms for processing skyline and reverse skyline queries in MapReduce; and in [13, 25] the problem of computing the skyline of a vast-sized spatial dataset in SpatialHadoop is studied. (4) *Reverse Nearest Neighbor (RNN) query* [21], where input data is partitioned

[1] `http://spark.apache.org/`

by a Voronoi diagram to exploit its properties to answer RNN queries. (5) *Spatial join* [12, 20, 22]; in [20] the *partition-based spatial-merge* join [23] is ported to MapReduce, and in [12] the *map* function partitions the data using a grid while the *reduce* function joins data in each grid cell. (6) *KNN join* [4, 5, 22], where the main target is to find for each point in a set P, its KNN points from set Q using MapReduce. (7) in [7], the problem of the *top-K closest pair problem* (where just one dataset is involved) with Euclidean distance using MapReduce is studied. (8) *KCP query* [6], the problem of answering the KCPQ (using plane-sweep techniques [28]) in SpatialHadoop is studied and evaluated. Finally, (9) the *similarity join* in high-dimensional data using MapReduce has been actively studied, the most representative work is [8], where a partition-based similarity join for MapReduce is proposed (called $MRSimJoin$). This approach is based on the $QuickJoin$ algorithm [26], and iteratively partitions the data until each partition can be processed on a single reducer (i.e. it partitions and distributes the data until the subsets are small enough to be processed in a single node).

DRQs have received considerable attention from the spatial database community, due to its importance in numerous applications [1]. SpatialHadoop is equipped with several spatial operations, including window query, KNN and spatial join [12], and other fundamental computational geometry algorithms as polygon union, skyline, convex hull, farthest pair, and closest pair [28]. And recently, new algorithms for skyline query processing have been also proposed in [13] and for KCPQ in [6]. And based on the previous observations, efficient processing of *DRQs* over large-scale spatial datasets using SpatialHadoop is a challenging task, and it is the main motivation of this paper.

3 Preliminaries and Background

In this section, we first present the basic definitions of the *DRQs*, followed by a brief introduction of preliminary concepts of SpatialHadoop, which is a full-fledged MapReduce framework with native support for spatial data.

3.1 Distance Range Queries

εDistance Range Query The ε Distance Range Query (εDRQ) reports all spatial objects from a spatial objects dataset that fall on the distance range defined by $[\varepsilon_1, \varepsilon_2]$ with respect to a query object. That is, it finds all spatial objects form the spatial dataset that are contained with the *annulus* centered in the query point with radios ε_1 and ε_2 (i.e. the region in the plane between two concentric circles of different radius). It is a special case of *Regional Query* [1], which permits search regions to have arbitrary orientations and shape. In our case, the region query is defined by a point query and an interval of distances, generating different circular shapes (e.g. if $\varepsilon_1 = 0$ and $\varepsilon_2 > 0$, then the region is a circle; if $\varepsilon_1 = \varepsilon_2 > 0$, then the region is a circumference; if $\varepsilon_1 = \varepsilon_2 = 0$, then the spatial objects intersect or they can be identical; etc.). The formal definition of εDRQ for point datasets (the extension of this definition to other complex spatial objects is straightforward) is the following:

Definition 1. (ε **Distance Range Query,** εDRQ)
Let $P = \{p_0, p_1, \cdots, p_{n-1}\}$ a set of points in E^d, a query point q in E^d, and a distance range defined by $[\varepsilon_1, \varepsilon_2]$ such that ε_1, $\varepsilon_2 \in \mathbb{R}^+$ and $\varepsilon_1 \leq \varepsilon_2$. Then, the result of the ε Distance Range Query with respect to the query point q is a set $\varepsilon DRQ(P, q, \varepsilon_1, \varepsilon_2) \subseteq P$, which contains all points $p_i \in P$ that fall on the circular shape, centered in q with radios ε_1 and ε_2:

$$\varepsilon DRQ(P, q, \varepsilon_1, \varepsilon_2) = \{p_i \in P \ : \ \varepsilon_1 \leq \ dist(p_i, q) \ \leq \ \varepsilon_2\}$$

An example of this spatial query could be to find all fitness centers between 1 and 2 kilometers from my home. This spatial query has been studied in centralized environments, however, when the dataset resides in a parallel and distributed framework, it has not been given enough attention.

εDistance Range Join Query The ε Distance Range Join Query ($\varepsilon DRJQ$) reports all the possible pairs of spatial objects from two different spatial objects datasets, having a distance between ε_1 and ε_2 of each other. $\varepsilon DRJQ$ is a generalization of the ε Distance Join Query (εDJQ), which is characterized by two spatial datasets and a distance threshold, ε, and it permits search pairs of spatial objects from two input datasets that are within distance ε from each other. In our case, the distance threshold is a range defined by an interval of distances $[\varepsilon_1, \varepsilon_2]$, and if $\varepsilon_1 = 0$ and $\varepsilon_2 > 0$, then we have the definition of εDJQ. Moreover, if $\varepsilon_1 = \varepsilon_2 = 0$, then we have the condition of Spatial Intersection Join, which retrieves all different intersecting spatial object pairs from two distinct spatial datasets [1]. This query is also related to the Similarity Join [8], where the problem of deciding if two objects are similar is reduced to the problem of determining if two high-dimensional points are within a certain distance threshold ε of each other.

Definition 2. (ε **Distance Range Join Query,** $\varepsilon DRJQ$)
Let $P = \{p_0, p_1, \cdots, p_{n-1}\}$ and $Q = \{q_0, q_1, \cdots, q_{m-1}\}$ be two set of points in E^d, and a distance range defined by $[\varepsilon_1, \varepsilon_2]$ such that ε_1, $\varepsilon_2 \in \mathbb{R}^+$ and $\varepsilon_1 \leq \varepsilon_2$. Then, the result of the ε Distance Range Join Query is a set $\varepsilon DRJQ(P, Q, \varepsilon_1, \varepsilon_2) \subseteq P \times Q$, which contains all the possible different pairs of points that can be formed by choosing one point $p_i \in P$ and one point $q_j \in Q$, having a distance between ε_1 and ε_2 of each other:

$$\varepsilon DRJQ(P, Q, \varepsilon_1, \varepsilon_2) = \{(p_i, q_j) \in P \times Q \ : \ \varepsilon_1 \leq \ dist(p_i, q_j) \ \leq \ \varepsilon_2\}$$

An example scenario of this spatial distance-based join query could be the following. Suppose we want to buy a house with convenient living surroundings. We find a collection of possible houses P and a collection of shopping centers as Q, with the units of x and y axis being in kilometers (km). Suppose that we wants to consider only houses with their distances to shopping centers being at most 1500 meters. So we issues an $\varepsilon DRJQ$ with $\varepsilon_1 = 0km$ and $\varepsilon_2 = 1.5km$. As the εDRQ, this spatial query has been also studied in centralized environments, but when the datasets reside in a parallel and distributed framework, it has not attracted similar attention. For this reason, the main target of this work is to study these spatial query in SpatialHadoop.

3.2 SpatialHadoop

SpatialHadoop [12] is a full-fledged MapReduce framework with native support for spatial data. Notice that MapReduce [11] is a scalable, flexible and fault-tolerant programming framework for distributed large-scale data analysis. A task to be performed using the MapReduce framework has to be specified as two phases: the *Map* phase is specified by a *map function* takes input (typically from Hadoop Distributed File System (HDFS) files), possibly performs some computations on this input, and distributes it to worker nodes; and the *Reduce* phase which processes these results as specified by a *reduce function*. An important aspect of MapReduce is that both the input and the output of the *Map* step are represented as *Key/Value pairs*, and that pairs with same key will be processed as one group by the *Reducer*: $map : (k_1, v_1) \rightarrow list(k_2, v_2)$ and $reduce : k_2, list(v_2) \rightarrow list(v_3)$. Additionally, a *Combiner function* can be used to run on the output of *Map* phase and perform some filtering or aggregation to reduce the number of keys passed to the *Reducer*.

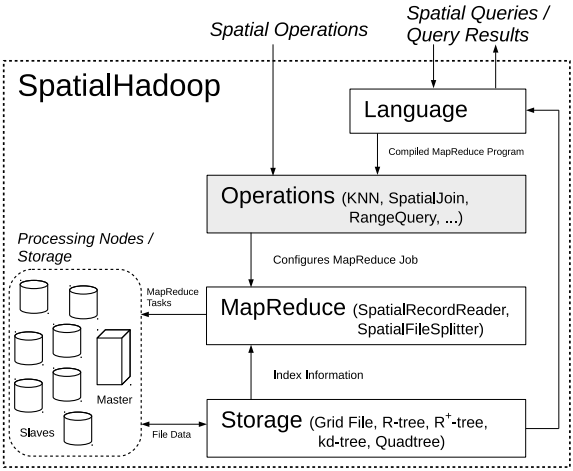


Fig. 1. SpatialHadoop system architecture [12].

SpatialHadoop, see in Figure 1 its architecture, is a comprehensive extension to Hadoop that injects spatial data awareness in each Hadoop layer, namely, the language, storage, MapReduce, and operations layers. In the *Language* layer, SpatialHadoop adds a simple and expressive high level language for spatial data types and operations. In the *Storage* layer, SpatialHadoop adapts traditional spatial index structures as Grid, R-tree and R^+-tree, to form a two-level spatial

index [27]. SpatialHadoop enriches the *MapReduce* layer by new components to implement efficient and scalable spatial data processing. In the *Operations* layer, SpatialHadoop is also equipped with a several spatial operations, including range query, KNNQ and spatial join. Other computational geometry algorithms (e.g. polygon union, skyline, convex hull, farthest pair, and closest pair) are also implemented following a similar approach [27]. Moreover, in this context, [13] improved the processing of skyline query implemented in SpatialHadoop. Finally, we emphasize that our contribution (εDRQs as spatial operations) is located in the *Operations* layer, as we can observe in Figure 1 in the highlighted box.

Since our main objective is to include the *DRQs* into SpatialHadoop, we are interested in the MapReduce and operations layers. *MapReduce layer* is the query processing layer that runs MapReduce programs, taking into account that SpatialHadoop supports spatially indexed input files. And the *operation layer* enables the efficient implementation of spatial operations, considering the combination of the spatial indexing in the storage layer with the new spatial functionality in the MapReduce layer. In general, a spatial query processing in SpatialHadoop consists of four steps: (1) *Partitioning*, where the data is partitioned according to a specific spatial index. (2) *Pruning*, when the query is issued, where the master node examines all partitions and prunes those ones that are guaranteed not to include any possible result of the spatial query. (3) *Local spatial query processing*, where a local spatial query processing is performed on each non-pruned partition in parallel on different machines. And finally, (4) *Global processing*, where a single machine collects all results from all machines in the previous step and compute the final result of the concerned query. And we are going to follow this query processing schema to include the *DRQs* (εDRQ and $\varepsilon DRJQ$) into SpatialHadoop.

4 *DRQ* Algorithms in SpatialHadoop

4.1 εDRQ Algorithm in SpatialHadoop

In this section, we describe our approach to the εDRQ algorithm on top of SpatialHadoop. In general, our solution is similar to how *range query* algorithm [12] is performed in SpatialHadoop, except instead of having a rectangular region (window), now we have a circular region defined by the query point and the range of distances $[\varepsilon_1, \varepsilon_2]$.

Firstly, we can use these ε_1, ε_2 values in combination with the features of indexing that are provided by SpatialHadoop to further enhance the pruning phase. Before the *Map* phase begins, we exploit the indexes to prune cells that cannot contribute to the final result. We partition the point dataset by some method (e.g. Grid or Str [27]) into blocks of points. CELLSFILTER algorithm takes as input each block / cell in which the input set of points is partitioned. Using SpatialHadoop built-in function *minDistance* we can calculate the minimum distance between a cell and the query point. That is, if we find a block with points which cannot have a distance value smaller than ε_2, we can prune

that block. Furthermore, we can use *maxDistance* to prune blocks with points which have distance values smaller than ε_1.

Once the filter is done, the answer is refined in the *Map* phase. In order to do this, a full-scan Algorithm 1 is implemented, where the set of points P is scanned giving points which satisfy the distance value restrictions ε_1 and ε_2 over the query point q as solutions. Notice that in the *Reduce* phase the results are just combined.

Algorithm 1 εDRQ MapReduce Algorithm

```
1: function MAP(P: set of points, q: query point, ε1: lb distance, ε2: ub distance)
2:     for all point ∈ P do
3:         distance ← DISTANCE(point, q)
4:         if distance ≤ ε2 then
5:             if distance ≥ ε1 then
6:                 OUTPUT(null, point)
7:             end if
8:         end if
9:     end for
10: end function
```

4.2 $\varepsilon DRJQ$ Algorithm in SpatialHadoop

In this section, we describe our approach to the $\varepsilon DRJQ$ algorithm on top of SpatialHadoop. This can be described as a special case of the KCPQ MapReduce job [6], where only we select the pairs of points in the range of distances $[\varepsilon_1, \varepsilon_2]$ for the final result. And the distance threshold will be always ε_2 instead of the distance of the K-th closest pair found so far [28]. Moreover, our approach adapts the two distance-based plane-sweep KCPQ algorithms (Classic and Reverse Run) for main-memory resident datasets [28] to $\varepsilon DRJQ$.

In [28], the *Classic Plane-Sweep* for KCPQ was reviewed and two new improvements were also presented, when the point datasets reside in main memory. In general, if we assume that the two point sets are P and Q, the *Classic* plane-sweep algorithm consists of the two following steps: (1) sorting the entries of the two point sets, based on the coordinates of one of the axes (e.g. X) in increasing order, and (2) combine one point (*pivot*) of one set with all the points of the other set satisfying $point.x - pivot.x \leq \varepsilon_2$. The algorithm chooses the *pivot* in P or Q, following the order on the sweeping axis.

In [28], a new distance-based plane-sweep algorithm for KCPQ was proposed for *minimizing the number of distance computations*. It was called *Reverse Run* plane-sweep algorithm. It also follows the *sort-and-scan paradigm* and it is based on two concepts. First, every point that is used as a *reference* point forms a *run* with other subsequent points of the same set. A *run* is a continuous sequence of points of the same set that doesn't contain any point from the other set. And second, the *reference* points (and their *runs*) are processed in ascending X-order (the sets are X-sorted before the application of the algorithm). Each point of

the *active run* is compared with the points of the other set (*comparison* points) in the opposite or reverse order (descending X-order) and the pair of points is selected if its distance is smaller than or equal to ε_2.

The two improvements presented in [28], called *sliding window* and *sliding semi-circle*, can be applied both *Classic* and *Reverse Run* plane-sweep algorithms. For the *sliding window*, the general idea consists of restricting the search space to the closest points inside the window with width ε_2 and a height $2 * \varepsilon_2$ (i.e. $[0, \varepsilon_2]$ in the X-axis and $[-\varepsilon_2, \varepsilon_2]$ in the Y-axis, from the *pivot* or the *reference* point). And for the *sliding semi-circle* improvement, it consists of trying to reduce even more the search space, since we can only select those points inside the semi-circle centered in the *pivot* or in the *reference* point with radius ε_2.

Algorithm 2 εDRJQ MapReduce Algorithm

```
1: function MAP(P: set of points, Q: set of points, ε1: lb distance, ε2: ub distance)
2:     SortX(P)
3:     SortX(Q)
4:     Results ← EDRJQ(P, Q, ε1, ε2)
5:     for all DistanceAndPair ∈ Results do
6:         output(DistanceAndPair.p, DistanceAndPair)
7:     end for
8: end function

9: function CELLSFILTER(C: set of cells, D: set of cells, ε1, ε2: lb and ub distances)
10:    for all c ∈ C do
11:        for all d ∈ D do
12:            minDistance ← MinDistance(c, d)
13:            if minDistance ≤ ε2 then
14:                maxDistance ← MaxDistance(c, d)
15:                if maxDistance ≥ ε1 then
16:                    output(c, d)
17:                end if
18:            end if
19:        end for
20:    end for
21: end function
```

The method for the εDRJQ in MapReduce is to adopt the *Map* phase join MapReduce methodology. The basic idea is to have P and Q partitioned by some method (e.g. Grid or Str [27]) into n and m blocks of points, respectively. Then, every possible pair of blocks (one from P and one from Q) is sent as the input for the *Filter* phase. In the same way as it was done in 4.1 a CELLSFILTER function prunes pairs of blocks which have maximum and minimum distances between them that do not match ε_1 and ε_2 values.

On the *Map* phase each *mapper* reads its pair of blocks and performs a εDRJQ algorithm (*Classic* or *Reverse Run*) between the local P and Q in that

pair. That is, it computes the εDRJQ between points in the local block of P and in the local block of Q using a εDRJQ plane-sweep algorithm. To do so, each *mapper* sorts the local P and Q blocks in one axis (e.g., X axis in ascending order) and then applies a particular *EDRJQ* algorithm. The results from all *mappers* are just combined in the *Reduce* phase and written into HDFS files, storing only the pairs of points with distance in $[\varepsilon_1, \varepsilon_2]$, as we can see in Algorithm 2.

5 Experimentation

In this section we present the results of our experimental evaluation. We have used synthetic (Uniform) and real 2d point datasets to test our *DRQs* algorithms in SpatialHadoop. For synthetic datasets we have generated several files of different sizes using SpatialHadoop built-in uniform generator [12]. For real datasets we have used three datasets from OpenStreetMap[2] [12]: *BUILDINGS* which contains 115M points of buildings, *LAKES* which contains 8.4M points of water areas, and *PARKS* which contains 10M points of parks and green areas [12]. To study the scalability of the datasets, subsets have been created with sample ratios of 25%, 50% and 75% from the original dataset (100%). We have used two performance metrics, the execution time and the number of complete distance computations. All experiments are conducted on a cluster of 20 nodes on an OpenStack environment. Each node has 1 vCPU with 2GB of main memory running Linux operating systems and Hadoop 1.2.1.

ε Distance Range Queries (εDRQ). For the different experiments, the *query point* is located at the center of the Minimum Bounding Rectangles (*MBRs*) to which the different datasets are partitioned by the SpatialHadoop indexing method utilized. For example, the MBR of the synthetic datasets is [(0,0)-(1000000,1000000)], and the query point is at (500000,500000).

Our first experiment is to examine the effect of the data set size. As we expected for uniform datasets, the results the execution time are almost uniform, due to the fact that the number of blocks that pass the filtering phase is less than the number of *map* tasks. However, for the experiments with real datasets, the execution time varies due to the partitioning performed on the data, since in a grid-based partitioning, the number of cells depends on the size of the data and more or less blocks due to a different partitioning obtained. For example, the number of points to consider for datasets sampled at 50% and 100% ratio are the same and are more than at 25% and 75%. And as we expected, the number of items returned by the query increases by the same percentage as data grows.

The second experiment studies the effect of different spatial partitioning techniques included in SpatialHadoop (Grid and Str [27]) and ε value ($\varepsilon_1 = 0$ and varying $\varepsilon_2 = \varepsilon$). As shown in Figure 2 left graph, the choice of a partitioning technique does not greatly affect the execution time showing a similar behavior. Using *Grid* partitioned files, the execution time increases linearly until almost

[2] Available at `http://spatialhadoop.cs.umn.edu/datasets.html`

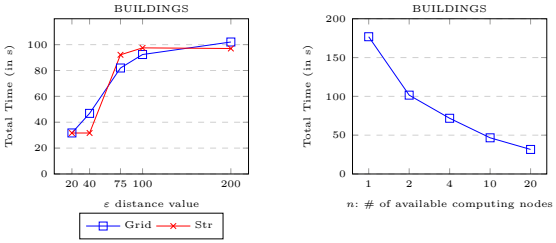


Fig. 2. Execution time vs. ε value (left) and execution time vs. n (right).

every point is selected and then it grows more slowly. As *Grid* partitioning is based on a uniform structure, the increment of ε values increases the number of selected blocks evenly. However, since *Str* partitioned files are nonuniform, the result is a stepped graph. For example, when ε value is 75, more blocks are selected and the execution time increases abruptly.

The third experiment aims to measure the speedup of the εDRQ algorithms, varying the number of computing nodes (n). Figure 2 right graph shows the impact of different numbers of computing nodes on the performance of parallel εDRQ algorithm. From this figure, it could be concluded that the performance of our approach has direct relationship with the number of computing nodes. It could be deduced that better performance would be obtained if more computing nodes are added. But, when the number of computing nodes exceeds the number of *map* tasks no improvement for that individual job is obtained.

ε Distance Range Join Queries (εDRJQ). The first experiment studies the effect of different spatial partitioning techniques included in SpatialHadoop (Grid and Str [27]). As shown in Figure 3 left graph, the choice of a partitioning technique greatly affects the execution time showing improvements of about 50% when using *Str* instead of *Grid*. Using *Grid* partitioned files, we get 175 combinations of blocks from input datasets, while using *Str* partitioned files just 79 combinations are obtained. This experiment is also useful to measure the scalability of the εDRJQ algorithms, varying the dataset sizes. We can observe that the execution time of our approach increases linearly.

The second experiment aims to find which of the different distance-based plane-sweep εDRJQ algorithms has the best performance. The execution times obtained do not show significant improvements between the different algorithms. This is due to various factors such as reading disk speed, network delays, the time for each individual task, etc. The metric shown in Figure 3 right graph is based on the number of times the algorithm performs a full calculation of the distance between two points. As shown in the right graph, any improvement (sliding Window or sliding Semi-Circle) on the *Classic* or *Reverse Run* algorithm obtains a much smaller number of calculations. The difference between these is

not very noticeable being the *Semi-Circle Reverse Run* algorithm the one with better results in most of the cases.

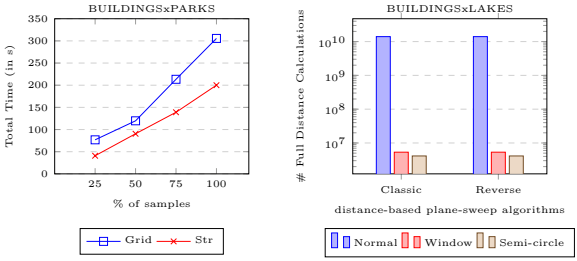


Fig. 3. Execution time vs. spatial partitioning techniques (left) and Number of complete distance computation vs. εDRJQ algorithm (right).

The third experiment studies the effect of the increasing of the ε value ($\varepsilon_1 = 0$ and varying $\varepsilon_2 = \varepsilon$). The scale of these ε values is different to that of the εDRQ (see Figure 2 left graph) due to the nature of the query. As show on Figure 4 left graph, the total execution time grows almost linearly as the ε value increases. It could be concluded that there is no real impact on the execution time but it must be taken into account that a higher ε value, the greater the possibility that pairs of blocks are not pruned and more *map* tasks could be needed.

The fourth experiment aims to measure the speedup of the εDRJQ algorithms, varying the number of computing nodes (n). Figure 4 right graph shows the impact of different node numbers on the performance of parallel εDRJQ algorithm, a behaviour similar to that of the εDRQ algorithm.

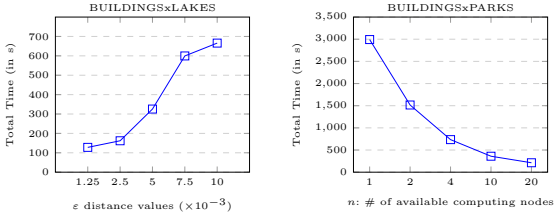


Fig. 4. Execution time vs. ε value (left) and execution time vs. n (right).

Conclusions from the experiments

– We have demonstrated experimentally the efficiency (in terms of total execution time and number of distance computations) and the scalability (in

terms of ε values, sizes of datasets and number of computing nodes) of the proposed parallel εDRQ and εDRJQ algorithms.
- Both plane-sweep-based algorithms (*Classic* and *Reverse Run*) used in the MapReduce implementation have similar performance in terms of execution time, although the *Semi-Circle Reverse Run* algorithm reduces slightly the number of complete distance computations.
- The use of a spatial partitioning technique included in SpatialHadoop [27] as *Str* (instead of *Grid*) improves notably the efficiency of the parallel εDRJQ algorithm. This is due to the fact that this variant organizes all partitions according to an R-tree structure at root level.

6 Conclusions and Future Work

DRQs (εDRQ and εDRJQ) are operations widely adopted by many spatial and GIS applications. They are characterized by a *filter* using a *distance range* over one or two datasets. These spatial queries have been actively studied in centralized environments, however, for parallel and distributed frameworks they have not attracted similar attention. For this reason, in this paper, we studied the problem of answering the DRQs in SpatialHadoop, an extension of Hadoop that supports spatial operations efficiently. To do this, we have proposed new parallel algorithms in MapReduce for the εDRQ and εDRJQ on big spatial datasets, adopting the plane-sweep methodology. We have also improved these MapReduce algorithms with the features of indexing that SpatialHadoop provides to further enhance the pruning phase. The performance of the algorithms in various scenarios with big synthetic and real-world points datasets has been also evaluated. And, the execution of such experiments has demonstrated the efficiency (in terms of total execution time and number of distance computations) and scalability (in terms of ε values, sizes of datasets and number of computing nodes) of our proposal. Future work might cover studying of *DRQs* with other partitioning techniques not included in SpatialHadoop and conducting tests to check the performance of the solution on clusters using the lastest version of Hadoop that can take advantage of new technologies such as YARN.

References

1. S. Shekhar and S. Chawla: "*Spatial databases - a tour*", Prentice Hall, 2003.
2. S. Zhang, J, Han, Z. Liu, K. Wang and Z. Xu: "SJMR: Parallelizing spatial join with MapReduce on clusters", *CLUSTER Conf.*, pp. 1-8, 2009.
3. S. You, J. Zhang and L. Gruenwald: "Spatial join query processing in cloud: Analyzing design choices and performance comparisons", *ICPP Conf.*, pp. 90-97, 2015.
4. C. Zhang, F. Li and J. Jestes: "Efficient parallel k-NN joins for large data in MapReduce", *EDBT Conf.*, pp. 38-49, 2012.
5. W. Lu, Y. Shen, S. Chen and B.C. Ooi: "Efficient processing of k nearest neighbor joins using MapReduce", *PVLDB* 5(10): 1016-1027, 2012.

6. F. García-García, A. Corral, L. Iribarne, M. Vassilakopoulos and Y. Manolopoulos: "Enhancing SpatialHadoop with Closest Pair Queries", *ADBIS Conf.*, *In press*, 2016.
7. Y. Kim and K. Shim: "Parallel top-K similarity join algorithms using MapReduce", *ICDE Conf.*, pp. 510-521, 2012.
8. Y.N. Silva and J.M. Reed: "Exploiting MapReduce-based similarity joins", *SIGMOD Conf.*, pp. 693-696, 2012.
9. J. Dean and S. Ghemawat: "MapReduce: Simplified data processing on large clusters", *OSDI Conf.*, pp. 137-150, 2004.
10. F. Li, B.C. Ooi, M.T. Özsu and S. Wu: "Distributed data management using MapReduce", *ACM Comput. Surv.* 46(3): 31:1-31:42, 2014.
11. C. Doulkeridis and K. Nørvåg: "A survey of large-scale analytical query processing in MapReduce", *VLDB J.* 23(3): 355-380, 2014.
12. A. Eldawy and M.F. Mokbel: "SpatialHadoop: A MapReduce framework for spatial data", *ICDE Conf.*, pp. 1352-1363, 2015.
13. D. Pertesis and C. Doulkeridis: "Efficient skyline query processing in SpatialHadoop", *Inf. Syst.* 54: 325-335, 2015.
14. J. Lu and R.H. Güting: "Parallel Secondo: Boosting database engines with Hadoop", *ICPADS Conf.*, pp. 738-743, 2012.
15. A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang and J.H. Saltz: "Hadoop-GIS: A high performance spatial data warehousing system over MapReduce", *PVLDB* 6(11): 1009-1020, 2013.
16. A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff and R. Murthy: "Hive - A warehousing solution over a MapReduce framework", *PVLDB* 2(2): 1626-1629, 2009.
17. S. You, J. Zhang and L. Gruenwald: "Large-scale spatial join query processing in cloud", *ICDE Workshops*, pp. 34-41, 2015.
18. J. Yu, J. Wu and M. Sarwat: "GeoSpark: A Cluster Computing Framework for Processing Large-Scale Spatial Data", SIGSPATIAL Conf., pp. 70-74, 2015.
19. Q. Ma, B. Yang, W. Qian and A. Zhou: "Query processing of massive trajectory data based on MapReduce", *CloudDB Conf.*, pp. 9-16, 2009.
20. S. Zhang, J. Han, Z. Liu, K. Wang and S. Feng: "Spatial queries evaluation with MapReduce", *GCC Conf.*, pp. 287-292, 2009.
21. A. Akdogan, U. Demiryurek, F.B. Kashani and C. Shahabi: "Voronoi-based geospatial query processing with MapReduce", *CloudCom Conf.*, pp. 9-16, 2010.
22. K. Wang, J. Han, B. Tu, J. Dai, W. Zhou and X. Song: "Accelerating spatial data processing with MapReduce", *ICPADS Conf.*, pp. 229-236, 2010.
23. J.M. Patel and D.J. DeWitt: "Partition based spatial-merge join", *SIGMOD Conf.*, pp. 259-270, 1996.
24. Y. Park, J.K. Min and K. Shim: "Parallel computation of skyline and reverse skyline queries using MapReduce", *PVLDB* 6(14): 2002-2013, 2013.
25. A. Eldawy, Y. Li, M.F. Mokbel and R. Janardan: "CG_Hadoop: computational geometry in MapReduce", *SIGSPATIAL Conf.*, pp. 284-293, 2013.
26. E.H. Jacox and H. Samet: "Metric space similarity joins", *ACM Trans. Database Syst.* 33(2): 1-38, 2008.
27. A. Eldawy, L. Alarabi and M.F. Mokbel: "Spatial partitioning techniques in SpatialHadoop", *PVLDB* 8(12): 1602-1613, 2015.
28. G. Roumelis, M. Vassilakopoulos, A. Corral and Y. Manolopoulos: "A new plane-sweep algorithm for the K-closest-pairs query", *SOFSEM Conf.*, pp. 478-490, 2014.

Hacia la Evaluación de Recomendadores Utilizando un Simulador de Entornos Móviles

Sergio Ilarri and Slavcho Ivanov

[1] Universidad de Zaragoza, I3A, Zaragoza, España
silarri@unizar.es
[2] Universidad de Zaragoza, Zaragoza, España
619885@unizar.es

Resumen Los sistemas de recomendación ofrecen recomendaciones personalizadas a usuarios acerca de ítems de distinto tipo (películas, libros, restaurantes, hoteles, lugares a visitar, etc.), aliviando así la sobrecarga de datos que estos experimentan cuando tienen que tomar decisiones al elegir entre diversas alternativas. Debido a su interés tanto para usuarios finales como para empresas, este tipo de sistemas han atraído una intensa actividad investigadora. En concreto, en los últimos años ha crecido el interés por los sistemas de recomendación dependientes del contexto y por su aplicación en escenarios de computación móvil.
Sin embargo, existen dificultades para evaluar las propuestas existentes debido a la carencia de conjuntos de datos apropiados para evaluación. En este artículo motivamos el interés de evaluar sistemas de recomendación mediante la realización de simulaciones para recoger datos y opiniones de usuarios reales. Asimismo, describimos las ideas principales detrás de la herramienta RecSim que hemos desarrollado.

Keywords: Sistemas de recomendación, Simulación, Computación Móvil

1. Motivación

Hoy en día los usuarios se ven sometidos a una gran sobrecarga de información, lo que hace que el proceso de decisión entre varias alternativas resulte un proceso difícil y con resultados inciertos. Por ello, los llamados *sistemas de recomendación* (*Recommender Systems* o *RS*) ofrecen recomendaciones personalizadas a los usuarios, lo que facilita su elección cuando están interesados en un determinado tipo de ítem (película, libro, restaurante, hotel, etc.). Estos sistemas tienen un gran interés desde el punto de vista económico, ya que permiten a las empresas hacer llegar a sus clientes recomendaciones de productos relevantes; como ejemplo, tres empresas conocidas que emplean sistemas de recomendación son Netflix, Amazon y Booking.com. Además, también resultan de utilidad para los usuarios, ya que actúan como filtro y ofrecen al usuario sólo información de aquello que puede encontrar de interés. No obstante, el diseño de sistemas de recomendación debe enfrentarse a dificultades como el problema del arranque en frío (*cold start problem*) o el problema de opiniones artificiales o manipuladas

(*spam*). En el extremo, si no se ofrecen buenas recomendaciones al usuario, esto llevará a una desensibilización del usuario ante el sistema de recomendación, que terminará por dejar de confiar en él y abandonarlo.

Las aproximaciones de recomendación tradicionales habitualmente consideran un modelo de dos dimensiones, *usuario x ítem* → *valoración*, que trata de estimar la valoración que el usuario daría a un ítem dado a partir de información de valoraciones realizadas por ese usuario y otros usuarios. Paradigmas habituales son el *filtrado colaborativo usuario-usuario* (recomendar a un usuario ítems que han gustado a usuarios con gustos similares), el *filtrado colaborativo ítem-ítem* (recomendar a un usuario ítems similares, en cuanto a las valoraciones recibidas de los diversos usuarios, a otros ítems que le han gustado en el pasado), y el *filtrado colaborativo basado en contenido* (recomendar a un usuario ítems similares, en cuanto a las características de dichos ítems, a otros ítems que le han gustado en el pasado). Por ejemplo, Booking.com ofrece recomendaciones del tipo *"Customers who viewed Hotel X also viewed..."* y *"Destinations related to Y"*. Además, en los últimos años, en la investigación en el área de los sistemas de recomendación se ha considerado la incorporación como tercera dimensión del contexto del usuario, dando lugar a los denominados *sistemas de recomendación dependientes del contexto* (*Context-Aware Recommender Systems* o *CARS*) [1], siguiendo un modelo *usuario x ítem x contexto* → *valoración*. La motivación es el interés de considerar el impacto que diversos parámetros del contexto del usuario (su localización, la actividad que está realizando, la hora del día, el tiempo atmosférico, etc.) pueden tener en la relevancia de ciertos ítems para el usuario en un momento dado. Además, hay intentos para tratar de unificar investigación realizada en el campo de gestión de datos en computación móvil con las aproximaciones existentes en el área de los sistemas de recomendación [3].

Sin embargo, la evaluación de sistemas de recomendación en entornos de computación móvil, donde hay usuarios móviles accediendo a las recomendaciones con sus dispositivos inalámbricos y donde además algunos ítems de interés podrían ser móviles (taxis, buses, personas, etc.), no está exenta de dificultades. Esto es debido a que los conjuntos de datos tradicionalmente utilizados para evaluar la calidad de los sistemas de recomendación no incorporan información acerca del contexto de los usuarios. Aunque existen algunos conjuntos de datos que incorporan atributos del contexto, como STS [2], la cantidad de información realmente utilizable es muy pequeña [3], ya que la mayoría de las variables del contexto del usuario cuando éste valora un ítem son desconocidas. Por ello, resulta muy relevante explorar otras posibilidades para evaluar estos sistemas. En particular, en este artículo presentamos los aspectos básicos de la herramienta RecSim, que permite simular entornos móviles y usuarios que valoran ítems en dichos entornos sin desplazarse de casa.

2. Funcionalidades Principales

El simulador RecSim permite representar *escenarios* adaptados a las necesidades de simulación existentes. Un escenario se compone de un mapa y de un

conjunto de objetos, y puede definirse sobre cualquier mapa real disponible en OpenStreetMap: el usuario puede utilizar un buscador para ver un mapa y luego seleccionar en dicho mapa el área geográfica de interés. Además, se pueden definir tipos de objetos estáticos (por ejemplo, restaurantes, hoteles, cines, etc.) y tipos de objetos dinámicos o móviles (por ejemplo, taxis, buses, ambulancias, etc.), asociándoles un nombre de tipo y un icono que se utilizará para representar instancias de dicho tipo de objeto en un mapa. Luego pueden colocarse en un escenario objetos estáticos y dinámicos de los tipos deseados: los objetos estáticos se colocan en la posición deseada sobre el mapa y a los objetos dinámicos se les asigna una determinada trayectoria o un comportamiento de movimientos aleatorios. Alternativamente, pueden importarse datos acerca de objetos estáticos o dinámicos definidos en ficheros externos. Sobre el mapa se representa también un usuario móvil, que puede desplazarse utilizando el teclado (sin necesidad de que el usuario real cambie físicamente de lugar) o bien desplazarse físicamente (en cuyo caso las coordenadas se obtienen de un receptor GPS).

Un aspecto clave es que podemos simular esos escenarios móviles y al mismo tiempo poner en funcionamiento sistemas de recomendación que proporcionan recomendaciones al usuario. El usuario puede valorar ítems, de forma que el sistema de recomendación puede ir aprendiendo los gustos del usuario y aumentando la información del conjunto de datos de valoraciones disponible. Como ejemplo, en la Figura 1 se muestra una captura de pantalla donde se puede ver un fragmento del mapa de Zaragoza con un usuario móvil y varios tipos de ítems de potencial interés: hoteles, restaurantes, hospitales, y coches. Es posible exportar resultados referentes a los errores cometidos por el sistema de recomendación al valorar ítems y generar gráficas sencillas desde el mismo simulador, que ilustran los errores cometidos por el sistema de recomendación al estimar la valoración de varios ítems. Es importante recalcar que el objetivo de este trabajo no es desarrollar sistemas de recomendación en sí, sino proporcionar un entorno de simulación que permita evaluar de la forma más sencilla posible cualquier sistema de recomendación deseado.

3. Arquitectura y Tecnologías Utilizadas

RecSim ha sido desarrollado utilizando tecnologías web y se accede a través de un navegador web, de forma que puede utilizarse tanto desde ordenadores personales como desde dispositivos móviles. Si se instala en un servidor, varios usuarios pueden acceder a él simultáneamente para realizar simulaciones a través de sus respectivos navegadores. Para el desarrollo del back-end se han utilizado Node.js, Express, socket.io y mongoose. El navegador web se conecta a un servidor Node.js mediante HTTP y mediante un sistema bidireccional dirigido por eventos. Funcionalidades como la creación de escenas y la búsqueda de mapas están desarrollados sobre un API REST con intercambio de mensajes JSON. El sistema bidireccional dirigido por eventos se utiliza durante la simulación tanto para reflejar los eventos generados por un usuario de forma que sean visibles por el resto de usuarios como para realizar la integración entre el navegador, el

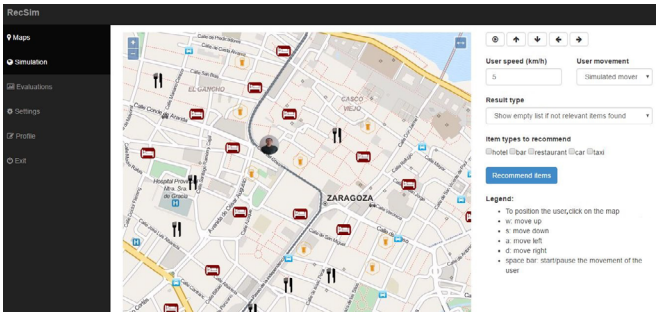


Figura 1. Captura del simulador en acción.

servidor Node.js y el recomendador. Para permitir una fácil integración de cualquier algoritmo de recomendación que se desee evaluar, se ha seguido el patrón de diseño *Strategy*.

4. Conclusiones y Trabajo Futuro

En este artículo se ha motivado el interés de utilizar simulaciones para evaluar sistemas de recomendación en entornos móviles y se han presentado las funcionalidades básicas ofrecidas por RecSim, un simulador diseñado y desarrollado para tal efecto. Como trabajo futuro se plantea extender el simulador con funcionalidades que permitan simular eventos dinámicos (por ejemplo, cambios en el tiempo atmosférico), así como la realización de pruebas con diversos usuarios y considerando distintos escenarios de ejemplo.

Acknowledgements

Trabajo financiado por el proyecto CICYT TIN2013-46238-C4-4-R y DGA-FSE.

Referencias

1. Adomavicius, G., Tuzhilin, A.: Context-aware recommender systems. In: Recommender Systems Handbook, pp. 217–253. Springer (2011)
2. Braunhofer, M., Elahi, M., Ricci, F., Schievenin, T.: Context-aware points of interest suggestion with dynamic weather data management. In: Information and Communication Technologies in Tourism, pp. 87–100. Springer (2013)
3. del Carmen Rodríguez-Hernández, M., Ilarri, S.: Pull-based recommendations in mobile environments. Computer Standards & Interfaces 44, 185–204 (February 2016)

Aproximación a la búsqueda basada en términos sobre conjuntos de datos medioambientales

David Álvarez-Castro, José R.R. Viqueira, and Alberto Bugarín

Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS)
Universidade de Santiago de Compostela
Santiago de Compostela, Spain
david.alvarez.castro@rai.usc.es, {jrr.viqueira, alberto.bugarin.diz.}@usc.es

Resumen En este artículo se discuten los trabajos, actualmente en curso, de diseño e implementación de un sistema de búsqueda por términos sobre fuentes de datos medioambientales, entre las que se incluyen fuentes de entidades geográficas y arrays que almacenan la variación espacio-temporal de distintas variables geo-físicas. Este tipo de sistemas facilitan el descubrimiento y el acceso a fuentes de datos de naturaleza científica a usuarios no expertos, que pueden utilizarlas en aplicaciones de muy diverso tipo.

Keywords: Búsqueda por términos, Datos Medioambientales, Recuperación de Información, Búsqueda Geoespacial

1. Introducción

Muchas disciplinas científicas necesitan para sus estudios datos sobre las condiciones cambiantes del medio ambiente. Un ejemplo de esta necesidad, en el área de la salud pública, puede ser el análisis del riesgo de aparición de epidemias de cólera [1], donde se plantea la localización de, por ejemplo, "Zonas de alta temperatura del agua de mar y elevada precipitación". Datos de este tipo, también combinados con otros datos de naturaleza geo-espacial, son de gran importancia para la toma de decisiones en muchas otras áreas como el turismo, en el que una necesidad de información posible sería: "Playas con poco oleaje y temperatura agradable cerca de algún punto de interés cultural". En la actualidad, sin embargo, la resolución de este tipo de necesidades de información requieren la intervención de expertos que conozcan la existencia, ubicación, disponibilidad y características detalladas de cada fuente de datos, así como los medios para acceder a los mismos.

Por su parte, los sistemas de búsqueda por términos han sido implementados con éxito para descubrir y acceder a fuentes de datos no estructuradas como la web. Recientemente, algunas aproximaciones proponen soluciones para la búsqueda por términos sobre fuentes de datos estructuradas, tanto sobre fuentes relacionales como de datos enlazados (Linked Data) [2,3]. Una gran parte de los

datos medioambientales disponibles no encajan en modelos basados en entidades como los anteriores, sino en modelos basados en estructuras de arrays multidimensionales con dimensiones espacio-temporales (datos raster). Por último, en algunas Infraestructuras de Datos Espaciales se proporcionan buscadores basados en términos sobre catálogos de metadatos [4]. Sin embargo, estos buscadores no permiten resolver las necesidades de información descritas anteriormente.

En este artículo se describen los trabajos, actualmente en curso, de diseño e implementación de una primera solución de búsqueda basada en términos para fuentes de datos medioambientales. Los términos que podrá usar el usuario en el sistema de búsqueda incluyen nombres de propiedades espaciales (*temperatura*, *oleaje*, *precipitación*), valores o términos lingüísticos imprecisos (*alto*, *bajo*, *poco*, *agradable*), nombres de entidades geográficas (*Santiago*, *Monasterio de Caaveiro*), nombres de tipos de entidades (*Hotel*, *Monasterio*, *Población*), relaciones espaciales (*cerca de*) y/o referencias a instantes e intervalos de tiempo.

2. Arquitectura del sistema

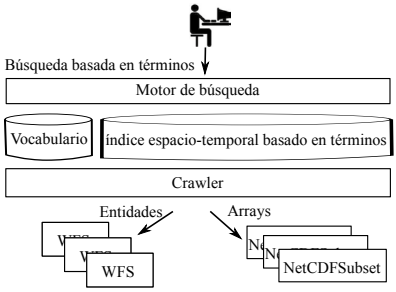


Figura 1. Arquitectura del sistema.

La arquitectura del sistema está basada en la arquitectura típica utilizada por los motores de búsqueda, tal y como se muestra en la Fig. 1. En la parte inferior de la figura se muestran los dos grandes tipos de fuentes de datos (entidades y arrays), que serán accedidos mediante estándares Web Feature Service (WFS) [5] y NetCDFSubset[1], bien conocidos y ampliamente utilizados. El *Crawler* descubre y accede a las fuentes de datos para actualizar el índice espacio-temporal basado en términos que permite responder a las búsquedas. Finalmente, en la

[1] https://www.unidata.ucar.edu/software/thredds/current/tds/reference/NetcdfSubsetServiceReference.html

parte superior de la arquitectura, el motor de búsqueda recibe las consultas basadas en términos (que pueden ser imprecisos) y utiliza el índice para generar la respuesta. Esta respuesta definirá de forma difusa las zonas del espacio y tiempo en las que se cumplen las condiciones especificadas, el grado de dicho cumplimiento (valor real en [0,1]) e incluirá referencias a las fuentes de datos utilizadas para su elaboración.

3. Subsistema de indexación

Para resolver expresiones del tipo "temperatura del agua moderada", el índice debe almacenar el grado de cumplimiento de dicha expresión en cada punto del espacio y del tiempo. Esta información se almacena de forma distinta en función de cómo dicha propiedad cambia en el espacio y tiempo.

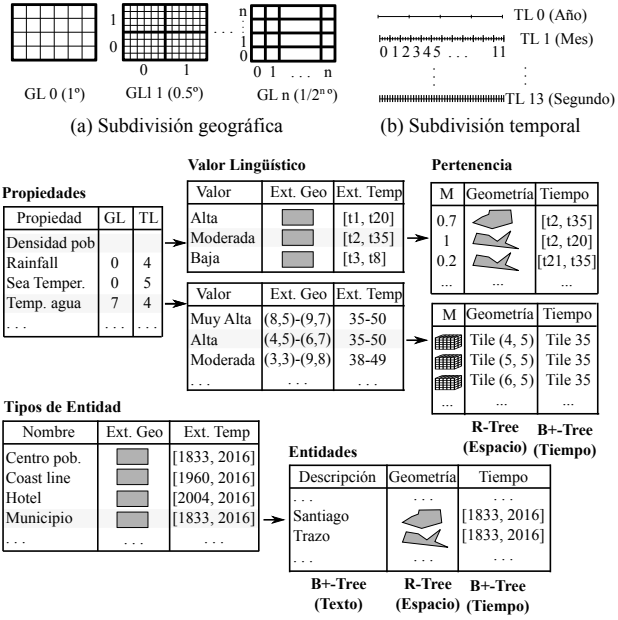


Figura 2. Índice espacio-temporal y baseado en términos

Si la propiedad cambia de forma continua, el cumplimiento para cada término difuso de la propiedad y para cada punto del espacio y tiempo se almacena en arrays tridimensionales. Para lograr una representación alineada en el espacio y el tiempo entre los datos generados de distintas fuentes se definen respectivas subdivisiones jerárquicas del espacio (ver Fig. 2(a)) y del tiempo (ver Fig. 2(b)). Como puede verse en la Fig. 2(c) para la propiedad "Temp. agua", se almacena el nivel en la jerarquía geográfica y temporal en la que se generaron los *tiles* de valores de pertenencia. Para cada etiqueta lingüística de cada propiedad se almacena el rango de *tiles* geográficos y temporales. Cada *tile* tendrá un array tridimensional de valores de pertenencia.

Si la propiedad cambia de forma discreta en el espacio y en el tiempo, se utiliza una representación espacial vectorial para las pertenencias, tal y como se puede ver en la Fig. 2(c) para la propiedad "Densidad pob".

Además de expresiones basadas en propiedades, el sistema permite también expresiones basadas en entidades geográficas y tipos, como por ejemplo "Cerca de Santiago" o "Lejos de un hotel". Para poder resolver estas expresiones, el índice almacena tanto tipos de entidades como entidades. Para cada entidad, además de su descripción textual, se guarda su geometría y tiempo de validez.

Para mejorar la eficiencia de acceso al disco, se utilizan estructuras de indexación para los textos, geometrías y marcas temporales.

4. Trabajo futuro

El trabajo futuro inmediato tiene que ver con la finalización de la implementación del primer prototipo y de su evaluación. A más largo plazo deberán abordarse nuevos retos relacionados con la mejora de la funcionalidad del sistema, su eficacia y su eficiencia. En este último caso será fundamental la incorporación de una arquitectura paralela de altas prestaciones.

Referencias

1. Baker-Austin, C., Trinanes, J.A., Taylor, N.G., Hartnell, R., Siitonen, A., Martinez-Urtaza, J.: Emerging vibrio risk at high latitudes in response to ocean warming. Nature Clim. Change 3(1), 73–77 (2013)
2. Bergamaschi, S., Guerra, F., Interlandi, M., Trillo-Lado, R., Velegrakis, Y.: Quest: A keyword search system for relational data based on semantic and machine learning techniques. Proc. VLDB Endow. 6(12), 1222–1225 (Aug 2013)
3. Demidova, E., Zhou, X., Nejdl, W.: A probabilistic scheme for keyword-based incremental query construction. IEEE Transactions on Knowledge and Data Engineering 24(3), 426–439 (March 2012)
4. Nebert, D., Whiteside, A., Vretanos, P.: OpenGIS Catalogue Services Specification. Open Geospatial Consortium (OGC) (2007), http://www.opengeospatial.org/standards/cat
5. Vretanos, P.: OpenGIS Web Feature Service 2.0 Interface Standard. Open Geospatial Consortium (OGC) (2010), http://www.opengeospatial.org/standards/wfs

A Federated Approach for Array and Entity Environmental Linked Data

Shahed Bassam Almobydeen, José R.R.Viqueira, and Manuel Lama Penín

Centro Singular de Investigación en Tecnoloxías da Información (CITIUS)
Universidade de Santiago de Compostela (USC)
Santiago de Compostela, Spain
{shahed.al-mobydeen, jrr.viqueira, manuel.lama}@usc.es

Abstract. This paper discusses the main challenges that arise during the design and implementation of a federated solution for entity and array based environmental linked data. The proposed solution enables the integrated querying of geospatial relational databases, large scientific arrays of spatio-temporal dimensions and linked data sources with GeoSPARQL. To achieve this, a query decomposition algorithm and two new operators have to be incorporated in an already existing SPARQL query engine.

Keywords: Linked data, Geospatial and environmental data, SPARQL Query processing

1 Introduction

Large amounts of environmental data are becoming available over the entire world in the form of open data repositories. Two major types of these geospatial data are available, namely *entity-based* and *field-based* data. The former fits conventional models and it is usually managed within spatially enabled databases. The latter is modeled as large arrays with spatial and temporal dimensions, and it is usually recorded either with specific scientific array formats or in array database technologies.

Most of the current environmental data consumers are experts of different scientific domains who have the required skills to discover, access and analyze them. However, many other applications could benefit from these data if they were appropriately accessible through standard linked data technologies [1]. One such example is Tourism, where already existing linked data repositories like DBPedia may be combined with geospatial *entity-based* data of locations, hotels, restaurants or site seeing and with meteorological predictions modeled as large spatio-temporal arrays (see Fig. 1).

A major challenge related to applications like the above one is the representation of very large spatio-temporal arrays in the RDF data model [3], which is the model for data representation in the linked data paradigm. Obviously, spatial and temporal dimensions, which are not explicitly recorded in scientific

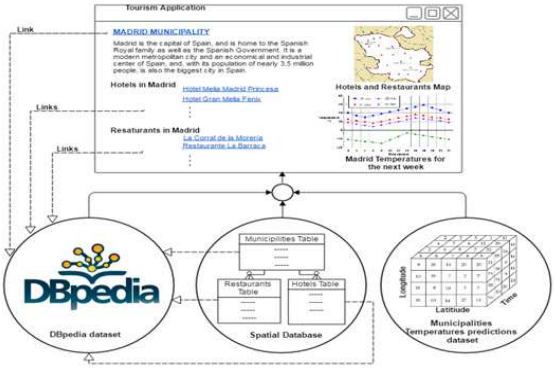


Fig. 1. Tourism Application.

array formats, must be explicitly recorded in traditional RDF encodings, leading this way to huge datasets that cannot be efficiently queried.

Many different solutions have been proposed during the last decade for the integrated querying of relational and linked data sources [6]. Some of them already considered datasets of *entity-based* geospatial data [2,4], which may currently be queried with the geographic extension of SPARQL (GeoSPARQL [5]). The above approaches may be classified according to their underlying data integration approach. Data warehouse approaches [4] use Extract Transform and Load (ETL) tasks to import relational spatial data into a spatially enabled RDF data storage technology. On the other hand, federated approaches [2] translate SPARQL queries to SQL to be evaluated directly by spatial relational databases. It is noticed that none of the above solutions has considered the incorporation of *field-based* large multidimensional array datasets.

A data warehouse solution would demand efficient storage formats and data access methods to integrate both entity and array-based RDF datasets. On the other hand, a federated approach, as the proposed in the present paper, requires the decomposition of SPARQL query algebra plans into three parts: one part to be translated to SQL, another one to be translated to some array query language, and a final part to be executed by the SPARQL engine by combining the above parts with native linked data sources.

2 Proposed federated approach

The main components of the architecture are (see Fig. 2):

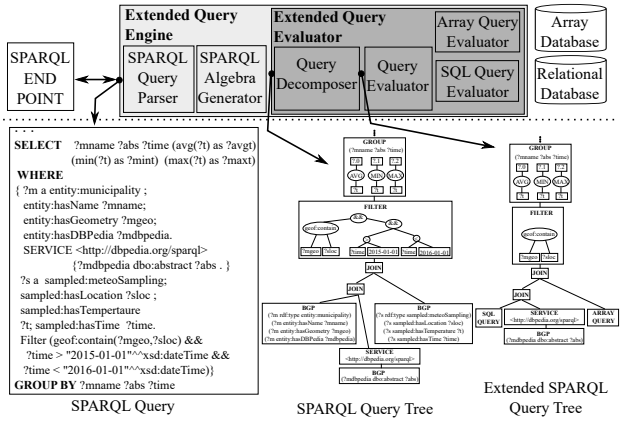


Fig. 2. Federated architecture.

- **SPARQL Query Parser**. GeoSPARQL queries submitted to the SPARQL end point are first parsed. The query illustrated in Fig. 2 retrieves the average, minimum and maximum predicted temperature for each municipality of DBPedia, at each time instant and for a specific time period.
- **SPARQL Algebra Generator**. A SPARQL query algebra tree is generated from the input parsed query. State of the art query optimization techniques are next applied. It is expected that filter conditions expressed using the vocabulary of just one data source are pushed down the tree close to the Basic Graph Pattern (BGP) operators that access the relevant data. In the above example the condition that restricts the time range in the array data source is pushed down to lay just above the BGP operator that access to the array data.
- **Query Decomposer**. This is the key component of the proposed solution. The optimized SPARQL query algebra tree is decomposed into three parts. The first one contains the maximum subtree that includes only vocabulary from the relational data source. In the above example, this is just the bottom left BGP operator that retrieves municipalities from the database. The second part contains the maximum subtree that includes only vocabulary from the array data source. In the above example, this is the bottom right BGP operator that retrieves temperature prediction, and the FILTER operator that restricts the search to a specific time period. The last part contains all the remainder nodes of the tree. The relational and array data source subtrees are next transformed to SQL and array query language, respectively. Those expressions are evaluated by two new SPARQL operators that gen-

erate result RDF triples from the retrieved relational and array data. The above operators are combined with the remainder part of the tree to yield a global SPARQL query algebra tree for the global query.

It is noticed that the proposed approach requires the implementation of a query decomposition algorithm and two new operators, one to execute SQL queries and another one to evaluate array queries. Both the query decomposition algorithm and the two operators are currently being implemented in the Apache Jena ARQ SPARQL engine.

3 Conclusion and future work

In this paper, we propose a federated architecture for accessing to entity and array environmental databases using linked data technologies. In our federated approach, SPARQL queries are used to access environmental data directly without loading them into RDF data stores. The new federated architecture enables expressing queries like "What is the predicted average of temperature of each municipality of Spain for the next week?". To achieve this, a query decomposition algorithm and two new opera-tors have to be developed. The implementation of the proposed federated architecture is still a work-in-progress. A comparison between the performance of the proposed federated architecture and already available data warehouse solutions will be done. Future work is mainly focused on the global query optimization at the mediator and on the incorporation of new Big Data storage and processing technologies.

References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. International Journal of Semantic Web Information Systems 5(3), 1–22 (2009)
2. Green, J., Hart, G., Dolbear, C., Engelbrecht, P.C., Goodwin, J.: Creating a semantic integration system using spatial data. In: Proceedings of the ISWC2008 Poster and Demonstration Session (2008)
3. Koubarakis, M.: Linked open earth observation data: The leo project. In: Proceedings of the The Image Information Mining Conference: The Sentinels Era (ESA-EUSC-JRC 2014). pp. 5–7. Bucharest, Romania (2014)
4. Kyzirakos, K., Karpathiotakis, M., Koubarakis, M.: Strabon: A semantic geospatial DBMS. In: The Semantic Web - ISWC 2012 - Proceedings of the 11th International Semantic Web Conference (ISWC2008). pp. 295–311 (2012)
5. Perry, M., Herring, J.: Ogc geosparql - A geographic query language for rdf data (2012)
6. Sahoo, S.S., Halb, W., Hellmann, S., Idehen, K., Thibodeau Jr, T., Auer, S., Sequeda, J., Ezzat, A.: A survey of current approaches for mapping of relational databases to rdf. Tech. rep., W3C RDB2RDF Incubator Group (2009), http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf

Procesamiento paralelo de datos medioambientales con Apache Spark

Diego Ferrón, Sebastián Villarroya, José R.R. Viqueira, and Tomás F. Pena

Centro de Investigación en Tecnoloxías da Información (CITIUS)
Universidade de Santiago de Compostela (USC)
Santiago de Compostela, Spain
diego.ferron@rai.usc.es, {sebastian.villarroya, jrr.viqueira, tf.pena}@usc.es

Resumen En este artículo se proporciona un descripción breve de los primeros pasos hacia la implementación de un sistema de procesamiento analítico en línea paralelo que integre datos de entidades espaciales y grandes arrays resultado de procesos de muestreo temporal, espacial y espacio-temporal. Estos tipos de dato son fundamentales en aplicaciones de apoyo a la toma de decisiones en entornos medioambientales.

Keywords: Big Data, Environmental Data, OLAP, Spark

1. Introducción

La importancia que tiene la incorporación de datos medioambientales, de naturaleza geo-espacial, en cada vez más ámbitos de toma de decisión es un hecho contrastado. La cantidad y precisión de los datos generados es cada vez mayor, debido a los grandes avances en la fabricación de dispositivos de sensorización. Así, por ejemplo, el análisis de los factores que influyen en la aparición de incendios forestales se beneficia de la disponibilidad de los siguientes conjuntos de datos (entre otros): i) Datos meteorológicos generados por redes de estaciones públicas y privadas. ii) Datos de elevación generados por sensores Lidar. iii) Datos de cobertura vegetal del terreno (modelos de combustible). Para hacerse una idea de la cantidad de datos generados, MeteoGalicia dispone de una red de unas 80 estaciones meteorológicas que generan datos con una frecuencia temporal de 10 minutos, mientras que la resolución espacial máxima de los datos de elevación proporcionados por el Instituto Geográfico Nacional es de 5 metros, con lo que se necesitan unos 2.500 millones de valores de elevación para cubrir el territorio de Galicia.

El reto es por lo tanto el procesamiento analítico en línea (OLAP) de grandes cantidades de datos que incluyen entidades espaciales (como las estaciones meteorológicas) y grandes arrays de valores numéricos resultantes de procesos de muestreo temporales (como los generados por las estaciones) y posiblemente también espaciales (como los generados por el Lidar). Así por ejemplo, el análisis de riesgo de aparición de incendios necesitaría: i) Calcular la pendiente del

terreno a partir de su elevación (operación de ventana espacial sobre los datos de elevación), ii) Interpolar espacialmente los datos meteorológicos de las estaciones para obtener una cobertura espacial completa (join espacial entre el muestreo espacial de elevación y las observaciones de las estaciones), iii) Combinación de la pendiente, datos meteorológicos y modelos de combustible para generar el índice de riesgo (join espacial).

El procesamiento de datos medioambientales en entornos científicos se realiza en general mediante implementaciones ad-hoc sobre formatos de archivo estandarizados, a veces utilizando librerías de geo-procesamiento disponibles en el área de los Sistemas de Información Geográfica. Si nos restringimos al procesamiento de datos de entidades espaciales, existen extensiones espaciales de los Sistemas Gestores de Bases de Datos más utilizados desde hace ya más de una década. En el rango del Big Data, existen también extensiones espaciales y geográficas para entornos de procesamiento bien conocidos, como Apache Hadoop [3] y Apache Spark [5]. Finalmente, existen también soluciones para el procesamiento de grandes arrays multidimensionales [1,2,6]. Sin embargo, no existe ninguna implementación eficiente de OLAP para datos medioambientales de todo tipo, diseñada para integrar entidades espaciales y grandes arrays resultantes de muestreos espacio-temporales.

En este artículo se ilustran los retos que surgen al abordar la implementación del lenguaje MAPAL (Mapping Analysis Language) [4] sobre una arquitectura de procesamiento paralelo de altas prestaciones basada en Apache Spark. Se discuten varias alternativas para el almacenamiento de datos y su influencia en la implementación de operaciones básicas de procesamiento.

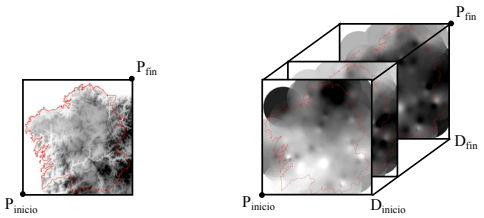
2. Solución propuesta

2.1. MAPAL: Lenguaje de análisis de funciones de datos

El sistema de tipos de dato de MAPAL [4] incluye tipos de dato convencionales, incluyendo numéricos de precisión fija y variable, tipos de dato temporales y espaciales (1D y 2D) de precisión fija y tipos de dato geométricos. Así, por ejemplo, el tipo de dato *TimeInstant*(R) representa instantes de tiempo con una resolución temporal de R segundos (R es un número real). El tipo *Point2D*(P,R) representa puntos de un espacio discreto de dimensión 2, donde cada punto se interpreta como un cuadrado (pixel) de tamaño real $R \times R$. Los tipos geométricos para representar líneas, polígonos y geometrías complejas se definen sobre la base del espacio definido por el tipo *Point2D*(P,R).

Los conjuntos de entidades espaciales y arrays se representan de forma uniforme mediante *Dimensiones* y *Cubos de Datos*. Una *Dimensión* es un subconjunto finito de los elementos de un tipo de dato (no geométrico). Casos especiales de *Dimensiones* son los *Muestreos 1D* (temporales y espaciales) y *2D* (espaciales). Un muestreo contiene una secuencia ordenada de elementos de un tipo de dato (temporal o espacial), desde un elemento inicial a un elemento final. Si las estaciones meteorológicas generan una serie de datos agregados de temperatura con

una frecuencia diaria, la *Dimensión* temporal de dicha serie se definiría como un *Muestreo 1D Temporal* sobre el tipo de datos *TimeInstant*(86400), desde una fecha inicial D_{inicio} a una fecha final D_{fin}. De forma similar, la *Dimensión* espacial para los datos de elevación se podría definir como un *Muestreo 2D* sobre el tipo de datos *Point2D*(7,5), desde un punto inicial (esquina inferior izquierda) P_{inicio} a un punto final (esquina superior derecha) P_{fin}.



(a) *Cubo de Datos* de Elevación (b) *Cubo de Datos* de Temperatura

Figura 1. Ilustración de *Cubos de Datos* espaciales y espacio-temporales.

Los *Cubos de Datos* (*MappingSets* en [4]) son estructuras n-dimensionales que permiten la representación de conjuntos de funciones, cuyos dominios están definidos por Productos Cartesianos de *Dimensiones* y cuyos rangos están definidos por subconjuntos de los tipos de dato del sistema. Por ejemplo, la elevación en cada punto se representa mediante un *Cubo de Datos* con una función *elevación* definida sobre el *Muestreo 2D* descrito arriba (ver Figura 1(a)). De forma similar, el resultado de la interpolación espacial de los datos generados por las estaciones se representa mediante un *Cubo de Datos* con tres funciones, *temperatura*, *humedad* y *viento*, de tipo numérico de precisión fija, definidas sobre el Producto Cartesiano de los muestreos temporal y espacial descritos arriba (ver Figura 1(b)).

2.2. Implementación sobre Apache Spark

En general tanto las *Dimensiones* como los *Cubos de Datos* se almacenan mediante *DataFrames*, persistidos mediante el formato columnar Apache Parquet. Tres alternativas distintas se están evaluando en la implementación en marcha.

Alternativa 1 Tanto las *Dimensiones* como las funciones se almacenan en columnas de un *DataFrame*.

Alternativa 2 Sólo las funciones se persisten en los *DataFrames*, ordenadas de forma ascendente por los valores de las *Dimensiones*.

Alternativa 3 Ademas de las funciones, se almacena un identificador autoincremental siguiendo el orden de las *Dimensiones*.

En cuanto al procesamiento, las operaciones entre *Dimensiones* y *Cubos de Datos* se implementan mediante operaciones de Apache Spark, que incluyen operadores relacionales sobre *DataFrames*. Una operación importante es el *equijoin* entre dos *Cubos de Datos* a través de *Dimensiones* comunes. Por ejemplo, el *equijoin* entre los *Cubos de Datos* de las Figuras 1(a) y (b) a través de su *Dimensión* espacial común. Dependiendo de la alternativa de almacenamiento utilizada para los *Cubos*, el algoritmo de este *equijoin* tendrá que leer más bloques de datos y realizar menos procesamiento (Alternativa 1) o leer menos bloques de datos y realizar más procesamiento (Alternativa 2). La Alternativa 3 es un compromiso entre las dos anteriores que facilita la generación de identificadores de *Dimensión* en un entorno de procesamiento paralelo.

Otro tipo importante de operación es el join espacial entre *Cubos de Datos*. En este caso, será fundamental el uso de una estrategia de particionamiento espacial de los datos para minimizar el tráfico entre los nodos.

3. Trabajo futuro

El trabajo futuro a más corto plazo consistirá en la finalización de la implementación en curso y su evaluación, incluyendo la evaluación del impacto en el rendimiento del uso de las tres alternativas de almacenamiento descritas y su combinación con distintas técnicas de codificación y compresión de datos. Posteriormente, se explorará el uso de distintas técnicas de particionamiento espacial que puedan ser utilizadas por nuevos algoritmos para los operadores de selección y join.

Referencias

1. Baumann, P., Dehmel, A., Furtado, P., Ritsch, R., Widmann, N.: The multidimensional database system rasdaman. In: Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data. pp. 575–577. SIGMOD '98, ACM, New York, NY, USA (1998)
2. Brown, P.G.: Overview of scidb: Large scale array storage, processing and analysis. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. pp. 963–968. SIGMOD '10, ACM, New York, NY, USA (2010)
3. Eldawy, A., Mokbel, M.F.: Spatialhadoop: A mapreduce framework for spatial data. In: 2015 IEEE 31st International Conference on Data Engineering. pp. 1352–1363 (April 2015)
4. Villarroya, S., Viqueira, J.R.R., Regueiro, M.A., Taboada, J.A., Cotos, J.M.: Soda: A framework for spatial observation data analysis. Distributed and Parallel Databases 34(1), 65–99 (2014)
5. Yu, J., Wu, J., Sarwat, M.: Geospark: A cluster computing framework for processing large-scale spatial data. In: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems. pp. 70:1–70:4. GIS '15, ACM, New York, NY, USA (2015)
6. Zhang, Y., Kersten, M., Manegold, S.: Sciql: Array data processing inside an rdbms. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. pp. 1049–1052. SIGMOD '13, ACM, New York, NY, USA (2013)

TINTIN: Comprobación Incremental de Aserciones SQL

Xavier Oriol[1], Ernest Teniente[1], and Guillem Rull[2] *

[1] Universitat Politècnica de Catalunya
{xoriol,teniente}@essi.upc.edu
[2] Universitat De Barcelona
grull@ceipac.ub.edu

Abstract. Ninguno de los SGBD actuales implementa aserciones SQL, obligando así a implementar manualmente su comprobación. Por este motivo hemos desarrollado TINTIN: una aplicación que genera automáticamente el código SQL necesario para comprobar aserciones. Dicho código captura las tuplas insertadas/borradas en una transacción, comprueba mediante consultas SQL que ninguna de ellas viole ninguna aserción, y materializa los cambios en caso de que esto suceda. La eficiencia del proceso se consigue mediante la comprobación incremental de las aserciones.

1 Introducción

Todo sistema de información tiene que poder capturar cualquier estado posible del dominio, y sólo los estados posibles de éste. Para conseguir dicho objetivo, los sistemas de información deben garantizar que sus datos siempre satisfacen ciertas condiciones, llamadas *restricciones de integridad* [1]. Por ejemplo, una restricción de integridad lógica en un sistema de información sobre libros, como el descrito en el diagrama UML de la Figura 1, sería que la fecha de publicación de toda edición de un libro tiene que ser posterior a la fecha de nacimiento de todos sus autores.

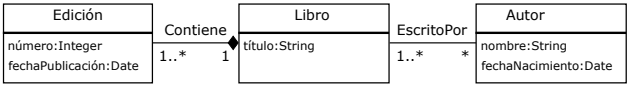


Fig. 1. Sistema de Información de libros y autores

Desde la publicación del estándar SQL-92, los sistema de información implementados con tecnología relacional pueden especificar sus restricciones de integridad mediante *aserciones* SQL. Intuitivamente, se puede definir una consulta SQL que recoja los datos que violan la restricción y una aserción que comprueba

* Este trabajo ha sido parcialmente financiado por el Ministerio de Economía y Competitividad, bajo el proyecto TIN2014-52938-C2-2-R; y la Secreteria d'Universitats i Recerca de la Generalitat de Catalunya bajo un 2014 SGR 1534 y una beca FI.

que el resultado de la consulta sea vacío. Según esta propuesta, la anterior restricción se podría definir como:

```
CREATE ASSERTION 'FechasCorrectas' CHECK NOT EXISTS (
        SELECT * FROM Edicion JOIN EscritoPor JOIN Autor
        WHERE fechaPublicacion <= fechaNacimiento)
```

Sin embargo, ninguno de los SGBD relacionales actuales (como por ejemplo Oracle, MySQL, SQL Server, PostgreSQL o DB2) implementa la comprobación automática de aserciones SQL. Por lo tanto, en a la práctica, dichas aserciones acaban siendo comprobadas mediante procedimientos programados manualmente, tarea laboriosa y propensa a errores [2]. En aras de superar esta limitación, en este artículo presentamos TINTIN [3]: una herramienta para generar automáticamente el código SQL necesario para comprobar aserciones.

2 El código generado por TINTIN

Supongamos que, en nuestro ejemplo, se añade el autor *Auguste Maquet* al libro *El Conde de Montecristo* que ya tenía como autor a *Alejandro Dumas*. Asumiendo que la aserción *FechasCorrectas* era cierta en el estado previo a esta inserción, podemos asegurar que después de ella lo seguirá siendo si la fecha de nacimiento de *Maquet* es anterior a la de todas las ediciones de *El Conde de Montecristo*, sin necesidad de comprobar los datos de otros libros/autores. A esta forma de comprobar aserciones en donde se presume que los datos actuales satisfacen las aserciones y se comprueba su validez únicamente con los nuevos datos modificados se le llama *comprobación incremental*.

TINTIN consigue una comprobación incremental mediante *triggers* que capturan los datos que están siendo insertados/borrados durante una transacción y los almacenan temporalmente en unas tablas con prefijo *ins/del*. De este modo, la inserción de *EscritoPor('El Conde de Montecristo','Auguste Maquet')* sería capturada y almacenada en la tabla *ins_EscritoPor* con los valores *('El Conde de Montecristo','Auguste Maquet')*, sin ser físicamente aplicada a la base de datos.

Teniendo capturadas las diferentes tuplas a insertar/borrar en las respectivas tablas *ins/del*, TINTIN puede generar unas consultas SQL que comprueban si estas entran en conflicto con alguna aserción. Siguiendo con el anterior ejemplo, TINTIN genera la siguiente consulta (almacenada como vista):

```
CREATE VIEW 'CheckFechasCorrectas'(
SELECT * FROM ins_Edicion JOIN EscritoPor ANTI JOIN del_EscritoPor JOIN Autor
WHERE fechaPublicacion <= fechaNacimiento
        UNION ALL
SELECT * FROM ins_EscritoPor JOIN Edicion ANTI JOIN del_Edicion JOIN Autor
WHERE fechaPublicacion <= fechaNacimiento
        UNION ALL
SELECT * FROM ins_EscritoPor JOIN Edicion ANTI JOIN del_Edicion JOIN
    ins_Autor
WHERE fechaPublicacion <= fechaNacimiento
        UNION ALL
SELECT * FROM ins_EscritoPor JOIN ins_Edicion JOIN Autor
WHERE fechaPublicacion <= fechaNacimiento
        UNION ALL
SELECT * FROM ins_EscritoPor JOIN ins_Edicion JOIN ins_Autor
WHERE fechaPublicacion <= fechaNacimiento
```

Intuitivamente, el primer *select* detecta las violaciones ocasionadas por inserciones de nuevas ediciones sin añadir nuevos autores. Cabe destacar aquí la necesidad de comprobar que el autor con el que se entra en conflicto no esté siendo borrado también por la transacción, hecho que se comprueba mediante una *anti join*. De manera similar, el segundo y tercer *select* detectan las violaciones ocasionadas al insertar nuevos autores a un libro sin añadir nuevas ediciones, y el cuarto y quinto detectan las violaciones ocasionadas por insertar nuevos autores y ediciones a un libro.

Una vez se tienen estas consultas almacenadas como vistas, sólo hace faltar invocarlas para saber si una transacción provoca o no la violación de alguna aserción. Ésta es precisamente la tarea desempeñada por el procedimiento *safeCommit*, también generado automáticamente por TINTIN. En concreto, *safeCommit* (1) ejecuta las vistas para comprobar si se viola alguna aserción, (2) en caso afirmativo, devuelve un mensaje de error al usuario indicando qué aserción está siendo violada y por qué datos; alternativamente, ejecuta los cambios almacenados en las tablas *ins/del*, (3) elimina el contenido almacenado en las tablas *ins/del* para así iniciar una nueva transacción desde cero. De este modo, la simple invocación de *safeCommit* al final de una transacción es suficiente para garantizar que los cambios definidos por dicha transacción sólo se realizarán si no violan ninguna aserción de la base de datos.

En la versión actual[3], TINTIN es capaz de generar el código para comprobar aserciones descritas en el fragmento de SQL equivalente al álgebra relacional.

3 Sobre cómo TINTIN genera el código

Presentamos ahora resumidamente cómo genera TINTIN las consultas SQL que detectan las violaciones.

En primer lugar, TINTIN reescribe las aserciones SQL como *denegaciones* lógicas siguiendo la traducción definida en [4]. Una denegación es una fórmula que establece una condición que nunca debe evaluarse a cierto en un estado de la base de datos. Por ejemplo, la anterior aserción se reescribiría como:

$$\texttt{edicion}(\texttt{n}, \texttt{fP}, 1) \wedge \texttt{escritoPor}(1, \texttt{a}) \wedge \texttt{autor}(\texttt{a}, \texttt{fN}) \wedge \texttt{fP} \leq \texttt{fN} \rightarrow \bot \quad (1)$$

A partir de aquí, TINTIN obtiene las correspondientes *Event Dependency Constraints* (EDCs) [5]. Cada EDC es una regla lógica que identifica una forma particular de violar una denegación mediante inserciones/borrados de tuplas sobre una base de datos D. Para obtener las EDCs, es suficiente con reemplazar cada literal en la denegación por la expresión que evalúa ese mismo literal en el nuevo estado de la base de datos D^n; o sea, el estado de la base de datos después de aplicar la transacción. En concreto, se reemplaza cada literal en función de si este es positivo o negativo mediante las siguientes equivalencias:

$$\forall \overline{x}.\ p^n(\overline{x}) \leftrightarrow (\iota p(\overline{x})) \vee (\neg\delta p(\overline{x}) \wedge p(\overline{x})) \quad (2)$$

$$\forall \overline{x}.\ \neg p^n(\overline{x}) \leftrightarrow (\delta p(\overline{x})) \vee (\neg\iota p(\overline{x}) \wedge \neg p(\overline{x})) \quad (3)$$

[3] `http://www.essi.upc.edu/~xoriol/tintin/`

donde $\iota p(\overline{x})/\delta p(\overline{x})$ representa la inserción/eliminación de $p(\overline{x})$. Así, según la regla 2, $p(\overline{x})$ es cierto en la nueva base de datos D^n si se inserta $p(\overline{x})$, o si $p(\overline{x})$ ya era cierto en D y no se elimina. La regla 3. define análogamente el caso $\neg p(\overline{x})$.

Utilizando estas sustituciones, en el anterior ejemplo se obtienen las reglas:

$$\texttt{ins\_ed(n,f,l)} \wedge \texttt{ins\_esc(l,a)} \wedge \texttt{ins\_aut(a,fN)} \wedge \texttt{f} \leq \texttt{fN} \rightarrow \bot \quad (4)$$

$$\texttt{ins\_ed(n,f,l)} \wedge \texttt{ins\_esc(l,a)} \wedge \texttt{aut(a,fN)} \wedge \neg\texttt{del\_aut(a)} \wedge \texttt{f} \leq \texttt{fN} \rightarrow \bot \quad (5)$$

$$\texttt{ins\_ed(n,f,l)} \wedge \texttt{esc(l,a)} \wedge \neg\texttt{del\_esc(l,a)} \wedge \texttt{ins\_aut(a,fN)} \wedge \texttt{f} \leq \texttt{fN} \rightarrow \bot \quad (6)$$

$$\texttt{ins\_ed(n,f,l)} \wedge \texttt{esc(l,a)} \wedge \neg\texttt{del\_esc(l,a)} \wedge \texttt{aut(a,fN)} \wedge \neg\texttt{del\_aut(a)} \wedge \texttt{f} \leq \texttt{fN} \rightarrow \bot \quad (7)$$

$$\texttt{ed(n,f,l)} \wedge \neg\texttt{del\_ed(n,l)} \wedge \texttt{ins\_esc(l,a)} \wedge \texttt{ins\_aut(a,fN)} \wedge \texttt{f} \leq \texttt{fN} \rightarrow \bot \quad (8)$$

$$\texttt{ed(n,f,l)} \wedge \neg\texttt{del\_ed(n,l)} \wedge \texttt{ins\_esc(l,a)} \wedge \texttt{aut(a,fN)} \wedge \neg\texttt{del\_aut(a)} \wedge \texttt{f} \leq \texttt{fN} \rightarrow \bot \quad (9)$$

$$\texttt{ed(n,f,l)} \wedge \neg\texttt{del\_ed(n,l)} \wedge \texttt{esc(l,a)} \wedge \neg\texttt{del\_esc(l,a)} \wedge \texttt{ins\_aut(a,fN)} \wedge \texttt{f} \leq \texttt{fN} \rightarrow \bot \quad (10)$$

$$\texttt{ed(n,f,l)} \wedge \neg\texttt{del\_ed(n,l)} \wedge \texttt{esc(l,a)} \wedge \neg\texttt{del\_esc(l,a)} \wedge \texttt{aut(a,fN)} \wedge \neg\texttt{del\_aut(a)} \wedge \texttt{f} \leq \texttt{fN} \rightarrow \bot \quad (11)$$

Si bien la cantidad de EDCs generada es exponencial con la longitud de la denegación, TINTIN incorpora ciertas optimizaciones para eliminar parte de estas reglas. Así, las EDCs 6 y 10 son eliminadas puesto que, para violarse, la base de datos debería violar primero la integridad referencial *escritoPor(autor)* $\rightarrow$ *autor(nombre)*. Adicionalmente, TINTIN elimina la EDC 11 ya que que únicamente se viola en una base de datos que previamente violara la misma denegación.

A partir de aquí, TINTIN traduce las EDCs a consultas SQL siguiendo las pautas establecidas en [6]. La validez y completitud de las vistas está garantizada puesto que el concepto de EDC se basa en las reglas de eventos [7], las cuales han sido demostradas como válidas y completas en bases de datos deductivas.

4 Conclusiones

Hemos presentado TINTIN, una herramienta para generar automáticamente el código SQL necesario para comprobar aserciones SQL. En el futuro, pretendemos extender TINTIN para (1) hacer frente a los agregados distributivos, (2) generar un número lineal de vistas extrayendo factor común entre las EDCs generadas.

Referencias

1. Olivé, A.: Conceptual Modeling of Information Systems. Springer, Berlin (2007)
2. Tropashko, V., Burleson, D.: SQL Design Patterns: Expert Guide to SQL Programming. Rampant Techpress (2007)
3. Oriol, X., Teniente, E., Rull, G.: TINTIN: a tool for incremental integrity checking of assertions in SQL server. In: Proceedings of the 19th Int. Conference on Extending Database Technology, EDBT. (2016) 632–635
4. Teniente, E., Farré, C., Urpí, T., Beltrán, C., Gañán, D.: SVT: schema validation tool for microsoft sql-server. In: Proc. of the 30th Int. Conference on Very Large Data Bases. (2004) 1349–1352
5. Oriol, X., Teniente, E., Tort, A.: Computing repairs for constraint violations in UML/OCL conceptual schemas. Data & Knowledge Engineering **99** (2015) 39 – 58
6. Oriol, X., Teniente, E.: Incremental checking of OCL constraints with aggregates through SQL. In: Conceptual Modeling. Volume 9381 of LNCS. Springer (2015) 199–213
7. Olivé, A.: Integrity constraints checking in deductive databases. In: Proceedings of the 17th Int. Conference on Very Large Data Bases (VLDB). (1991) 513–523

WikInfoboxer: A Tool to Create Wikipedia Infoboxes Using DBpedia

Ismael Rodriguez-Hernandez, Raquel Trillo-Lado, and Roberto Yus

University of Zaragoza, Zaragoza, Spain
{587429, raqueltl, ryus}@unizar.es

Abstract. Wikipedia infoboxes present a summary, in a semistructured format, of the articles they are associated to. Therefore, they have become the main information source used by projects to leverage the knowledge in Wikipedia, such as DBpedia. However, creating quality infoboxes is complicated as current mechanisms are based on simple templates which, for example, do not check whether the information provided is semantically correct.
In this paper, we present *WikInfoboxer*, a tool to help Wikipedia editors to create rich and accurate infoboxes. WikInfoboxer computes attributes that might be interesting for an article and suggests possible values for them after analyzing similar articles from DBpedia. To make the process easier for editors, WikInfoboxer presents this information in a friendly user interface.

Keywords: Infoboxes, Wikipedia, DBpedia, Semantic Web

1 Introduction

Nowadays, Wikipedia[1] is the biggest free and collaborative encyclopedia thanks to the collaboration of thousands of anonymous editors along the world. Thus, it is one of the most visited websites on the Web. Moreover, Wikipedia is also the main information source for other projects, such as DBpedia [1] or Google Knowledge Graph [3], which aim to automatically process data in order to speed up research studies and analytics. Indeed, these projects benefit from the semistructured information in the *infoboxes* in some Wikipedia articles. Infoboxes are a fixed-format table with partially established parameters that provides standardized information across related articles and summarizes the most relevant facts in them. So, the higher the quality of Wikipedia articles and their infoboxes, the more reliable the results obtained in the studies based on them.

Currently, if a Wikipedia editor wants to include infoboxes in an article, firstly, she must select the infobox templates (which are also Wikipedia entries) appropriate for that article. After the selection process, the templates must be copied in the article and the editor has to fill in the values of the attributes in the template (e.g., birth place, name) by taking into account the textual

[1] https://www.wikipedia.org

hints provided in the articles describing the templates. This procedure is prone-to errors as templates are free form in nature and there is not any check of the type of input data either recommendation of values to fill in the attributes. Moreover, the same attribute coming from different templates can be interpreted in different ways and be filled with different inconsistent values. Therefore, only a quarter of the articles in Wikipedia have infoboxes approximately and a lot of infoboxes contain inaccurate data[2].

In this paper, we present *WikInfoboxer*[3], a tool based on Infoboxer (a previous prototype which has been updated [5]), whose main goal is becoming part o MediaWiki[4] to help Wikipedia editors to create rich and accurate infoboxes. Thus, our tool: 1) provides editors with suggestions about the templates to be used to create the infoboxes associated to their articles in Wikipedia, 2) ranks and aggregates the attributes of those templates, 3) suggests values to fill them whenever possible, and 4) links them to existing entities in Wikipedia. These functionalities are supported by statistic and semantic information extracted from DBpedia and Wikipedia templates. The rest of the paper is structured as follows. In Section 2, the architecture of WikInfoboxer and the technologies used to implement it are presented. Moreover, how to use WikInfoboxer and how to integrate it in MediaWiki and other WikiMedia Foundation[5] projects is depicted. Finally, the scenarios that will be presented in the demonstration session and the future work are described in Section 3.

2 Technical Overview

WikInfoboxer is a Web Information System composed of several modules (see Figure 1) that uses both statistical and semantic knowledge from a Knowledge Base to identify: 1) the most relevant attributes for a given Wikipedia entity and 2) the most relevant values and types for those attributes. Moreover, it also guides editors on the process of the selection of infoboxes templates for a Wikipedia entity, avoiding users to fill inconsistent data when redundant attributes appear in several templates selected.

The *User Interaction Module* manages user interactions and visualizes data previously obtained from the web server of the back-end of the system through HTTP calls. It can work in two different modes according to the expertise of the users: 1) *Expert Mode*, displaying statistical and semantic information about the attributes and their use (see Figure 2.a), and 2) *Basic Mode*, hiding details about the features of the attributes and their values for non-experts (see Figure 2.b). This module has been developed by using AngularJS[6] and Bootstrap[7].

[2] Data obtained from `https://en.wikipedia.org/wiki/Wikipedia:WikiProject_Infoboxes/Statistics` accessed 25th April 2016.
[3] `http://sid.cps.unizar.es/Infoboxer`
[4] Software used to create and maintain Wikipedia (`https://www.mediawiki.org`).
[5] A non-profit organization that supports and operates Wikipedia and other free knowledge projects (`https://wikimediafoundation.org`).
[6] A framework to build Single Page Applications (`https://angularjs.org`).
[7] A framework to develop Web Responsive Applications (`http://getbootstrap.com`).

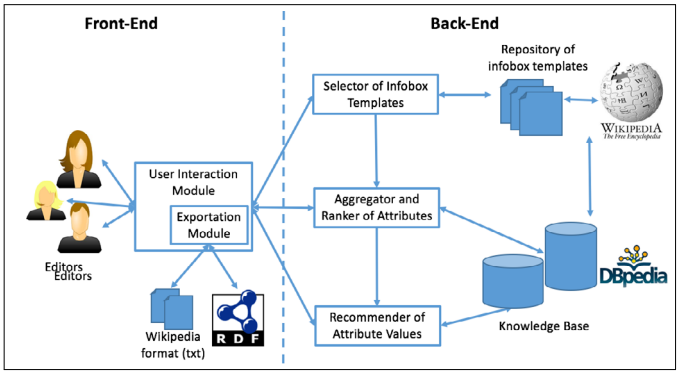


Fig. 1. High-level architecture of WikInfoboxer.

WikInfoboxer exports the created infoboxes in different formats. As the main goal is to help editors to create infoboxes for Wikipedia articles, the *Exportation Module* generates Wikipedia code in order to be directly copied and pasted in Wikipedia articles. Nevertheless, infoboxes are also translated into RDF to be loaded in the Knowlege Base. Moreover, the system can be easily adapted to export the information in the Wikidata format and publish it in Wikidata [4] by means of the Wikidata Toolkit[8].

The Back-End of WikInfoboxer is composed of three main modules (*Selector of Infobox Templates*, *Aggregator and Ranker of Attributes*, and *Recommender of Attribute Values*) in charge of selecting and computing statistical and semantics data, such as the domain and the range of properties, from Wikipedia and the *Knowledge Base (KB)* to guide editors in the process of creation of their infoboxes. It has been developed using the following technologies: Spring[9], OWL API[10] along with the HermiT reasoner[11], and a relational database management system (MySQL) to store statistics about users actions: filled properties, required timed, etc. and to maintain a cache of the results.

WikInfoboxer uses the DBpedia dump released in August 2015 as Knowledge Base. Nevertheless, other Open Knowledge Base or ontologies, such as Freebase or Wikidata, could be used. Performing efficient access to the Knowledge Base is essential to compute statistics for the different templates, attributes and values (and combining them) in real-time. So, the dump is managed by means of the

[8] An open source Java Library to interact with Wikidata (`https://www.mediawiki.org/wiki/Wikidata_Toolkit`).
[9] A framework to develop Java-based applications (`https://spring.io`).
[10] A Java API for OWL ontologies (`http://owlapi.sourceforge.net`).
[11] An OWL 2 reasoner (`http://www.hermit-reasoner.com`).

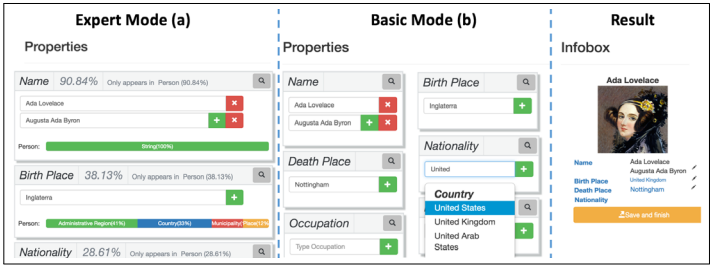


Fig. 2. WikInfoboxer Graphical User Interface.

SPARQL server Apache Jena Fuseki using HDT [2] (a compressed RDF format) to speed up the select operations.

3 Demonstration Highlights and Future Work

The demonstration will consist of two phases. Firstly, we will ask attendances to create an infobox by using the current interface of Wikipedia for that purpose. After that, they could try to create an analogous infobox by using WikInfoboxer. Moreover, technical details, and data and graphics about the performance of WikInfoboxer will be shown.

As future work, we plan to incorporate WikInfoboxer as plugin of MediaWiki and integrate it on Wikipedia (see the proposal available on `https://meta.wikimedia.org/wiki/Grants:IEG/WikInfoboxer`). Moreover, we would like to explore the possibilities of WikInfoboxer as tool to populate domain-ontologies in different contexts such as public administrations and research environments.

Acknowledgments. This research was supported by the CICYT project TIN-2013-46238-C4-4-R, DGA FSE, and Keystone Action COST IC1302.

References

1. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: DBpedia - a crystallization point for the web of data. Web Semantics: Science, Services and Agents on the World Wide Web 7(3), 154–165 (2009)
2. Fernández, J.D., Martínez-Prieto, M.A., et al.: Binary RDF representation for publication and exchange (HDT). Journal of Web Semantics 19, 22–41 (2013)
3. Singhal, A.: Introducing the knowledge graph: Things, not strings. In: Official Blog of Google (2012)
4. Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Communications of the ACM 57, 78–85 (2014)
5. Yus, R., Mulwad, V., Finin, T., Mena, E.: Infoboxer: Using statistical and semantic knowledge to help create Wikipedia infoboxes. In: 13th ISWC. pp. 405–408 (2014)

kpath: integration of metabolic pathway linked data

Ismael Navas-Delgado[1], María Jesús García-Godoy[1], Esteban López-Camacho[1], Maciej Rybinski[1], Armando Reyes-Palomares[2,3], Miguel Ángel Medina[2,3], and José F. Aldana-Montes[1]

[1] Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Andalucía Tech, Ada Byron Research Building, E-29071 Málaga, Spain
[2] Departamento de Biología Molecular y Bioquímica, Facultad de Ciencias, Universidad de Málaga, Andalucia Tech, and IBIMA (Biomedical Research Institute of Málaga), E-29071 Málaga, Spain
[3] CIBER de Enfermedades Raras (CIBERER) E-29071 Málaga, Spain

Resumen En los últimos años, el dominio de las Ciencas de la Vida ha experimentado un rápido crecimiento en la cantidad disponible de bases de datos biológicas. La heterogeneidad de estas bases de datos hace que la integración de datos sea difícil. Algunos retos de integración en este contexto son la localización de los recursos, descubrimiento de relaciones, heterogeneidad de formatos, definición de sinónimos y resolución de ambigüedad de los datos. El enfoque de Datos Vinculados (Linked Data) resuelve parcialmente los problemas de integración introduciendo un modelo de representación de los datos uniforme. Linked Data hace referencia a un conjunto de buenas prácticas para publicar y enlazar datos estructurados en la Web. Este artículo presenta *kpath* (http://sparql.kpath.khaos.uma.es/), una base de datos que integra información de rutas metabólicas. *kpath* también proporciona una interfaz visual que permite no solo la navegación, sino también el uso en profundidad de los datos integrados para construir redes metabólicas basadas en conocimiento disperso existente (http://browser.kpath.khaos.uma.es/). Esta interfaz de usuario ha sido utilizada para obtener relaciones que pueden ser inferidas a partir de la información disponible en varias bases de datos públicas.

Keywords: Linked Data, Metabolic Pathways, Data Integration

Practical Compressed String Dictionaries

Miguel A. Martínez-Prieto[1], Nieves Brisaboa[2], Rodrigo Cánovas[3], Francisco Claude[4], and Gonzalo Navarro[5]

[1] DataWeb Research, Universidad de Valladolid
migumar2@infor.uva.es
[2] Laboratorio de Bases de Datos, Universidade da Coruña
brisaboa@udc.es
[3] Department of Computing and Information Systems (CIS), The Univerity of Melbourne, Australia
rcanovas@student.unimelb.edu.au
[4] Escuela de Informática y Telecomunicaciones, Universidad Diego Portales, Chile
fclaude@recoded.cl
[5] Departmento de Ciencias de la Computación, Universidad de Chile, Chile
gnavarro@uchile.cl

Resumen. Este artículo afronta un problema recurrente en aplicaciones que gestionan colecciones de datos de diferente naturaleza: el almacenamiento y la consulta del conjunto de *strings* diferentes (vocabulario) utilizado en la colección. Las estructuras de datos que satisfacen estas necesidades se refieren, comúnmente, como **diccionarios**. Aunque la gestión de vocabularios no es un problema nuevo, sí lo es su explotación en escenarios en los que se utilizan grandes colecciones de datos. Por ejemplo, aplicaciones de procesamiento de lenguaje natural, basadas en los recursos de la web semántica, motores web o sistemas bioinformáticos, entre otros. En todos estos casos, el tamaño del vocabulario representa una fracción importante de la colección y el uso de las estructuras de diccionario habituales plantea un problema de escalabilidad por sí mismo.

Nuestro trabajo plantea una revisión de las soluciones utilizadas tradicionalmente para la implementación de diccionarios de texto (*hashing, tries* y codificación diferencial) y ofrece una mejora, para todas ellas, basada en el uso de diferentes técnicas de compresión. Además, proponemos varias técnicas innovadoras que se desarrollan sobre las emergentes estructuras de datos compactas y los (auto)índices *full-text*. El artículo también presenta una evaluación exhaustiva de todas nuestras propuestas en un entorno de experimentación heterogéneo que comprende vocabularios procedentes de múltiples escenarios de la vida real. Los resultados muestran como nuestras técnicas utilizan hasta 20 veces menos espacio que el ocupado por el propio vocabulario y soportan operaciones de acceso a los datos en el orden de unos pocos microsegundos. Estos números suponen una mejora competitiva respecto a los *tradeoffs* espacio/tiempo existentes en el estado del arte. Además, algunas de nuestras técnicas soportan operaciones de búsqueda avanzada (por prefijo y por *substring*), cuyos resultados también resultan altamente competitivos.

En definitiva, este trabajo muestra el beneficio de utilizar diccionarios de texto comprimidos y provee numerosas implementaciones que pueden utilizarse como bloques para la construcción de aplicaciones intensivas en datos.

XXI Jornadas de Ingeniería del Software y Bases de Datos, pp. 403-404

Publicación. El trabajo *Practical Compressed String Dictionaries* fue aceptado originalmente en Agosto de 2015 y publicado en el número 56 de la la revista Information Systems, en el presente 2016.

Information Systems es una revista publicada por la editorial Elsevier (ISSN 0306-4379) y está posicionada en el primer cuartil (Q1) de la categoría COMPUTER SCIENCE, INFORMATION SYSTEMS, correspondiente al *Journal Citation Report (JCR) ranking*. Su factor de impacto en 2015 fue de 1,832. La referencia completa al artículo es la siguiente:

- Miguel A. Martínez-Prieto, Nieves Brisaboa, Rodrigo Cánovas, Francisco Claude, and Gonzalo Navarro. Practical Compressed String Dictionaries. *Information Systems*, 56: 73–108, 2016.

Las Redes de Vehículos desde la Perspectiva de Gestión de Datos

Sergio Ilarri, Thierry Delot, and Raquel Trillo-Lado

[1] Universidad de Zaragoza, I3A, Zaragoza, España
silarri@unizar.es
[2] Universidad de Valenciennes, Valenciennes, France
tdelot@univ-valenciennes.fr
[3] Universidad de Zaragoza, I3A, Zaragoza, España
raqueltl@unizar.es

Resumen El interés por desarrollar sistemas de transporte inteligentes y redes de vehículos ad hoc se ha incrementado en los últimos años. Como bloque constructor fundamental para el desarrollo de aplicaciones para redes de vehículos, se necesitan nuevas técnicas para la gestión apropiada de datos en los vehículos. En este artículo, presentamos una extensa panorámica de la gestión de datos para redes de vehículos, donde las comunicaciones vehículo-a-vehículo juegan un papel clave. Describimos el contexto tecnológico de las redes de vehículos, así como los diferentes tipos de datos gestionados en dicho entorno, y analizamos varios desafíos, tales como la evaluación de la relevancia de los datos referentes a eventos que ocurren en las carreteras (por ejemplo, accidentes), el diseño de protocolos de diseminación de datos basados en contenido efectivos y eficientes, la competición en el acceso a recursos físicos (por ejemplo, plazas de aparcamiento), el desarrollo de técnicas de agregación específicamente adaptadas al contexto de redes de vehículos, y el procesamiento de consultas. El artículo proporciona una extensa cobertura de la gestión de datos para redes de vehículos, pero tiene al mismo tiempo una orientación didáctica. Respaldado por una extensa colección de referencias bibliográficas relevantes, analizamos el estado del arte, identificamos referencias de lectura obligada, esbozamos los problemas de investigación existentes, y extraemos conclusiones y lecciones aprendidas.

Keywords: Redes de vehículos, gestión de datos, diseminación de datos, compartición de datos, agregación de datos, procesamiento de consultas, recursos físicos escasos en las carreteras

Acknowledgements

Este trabajo ha sido financiado por el proyecto CICYT TIN2013-46238-C4-4-R y DGA-FSE.

Referencia Original

S. Ilarri, T. Delot and R. Trillo-Lado, "A Data Management Perspective on Vehicular Networks", IEEE Communications Surveys and Tutorials, ISSN 1553-877X, volume 17, number 4, IEEE Computer Society, 2420-2460, Fourthquarter 2015.
(DOI: 10.1109/COMST.2015.2472395)

Indicios de Calidad

JCR 2015: factor de impacto: 9,220; 1/143 en la categoría de *Computer Science, Information Systems*; 1/82 en la categoría de *Telecommunications*.

Validación y Diagnosis en tiempo de Ejecución sobre Reglas de Cumplimiento en Datos para Procesos de Negocio

María Teresa Gómez-López[1], Rafael M. Gasca[1], José Miguel Pérez-Álvarez[1]

[1] Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla,
Grupo IDEA (www.idea.us.es),
España
{maytegomez, gasca, josemi}@us.es

Abstract. Los procesos de negocio involucran datos que pueden ser creados, modificados y borrados por las diferentes actividades que forman dichos procesos de negocio. Estos datos pueden formar parte del flujo del proceso o ser almacenados en repositorios externos, típicamente bases de datos relacionales. La corrección de dichos datos es fundamental para el buen funcionamiento de los procesos, ya que ninguna actividad puede funcionar correctamente utilizando datos incorrectos. Hasta ahora, las propuestas de validación sobre datos se han centrado exclusivamente en los datos que fluyen en cada instancia, pero sin duda la información de fuentes externas también puede incluir en la ejecución de una instancia de proceso y tiene que ser correcta. Para la validación y la diagnosis de los datos, estos tienen que cumplir las reglas de cumplimiento definidas por el negocio, y activas en las diferentes actividades que se ejecutan en cada una de las instancias. En este trabajo se propone la creación de un framework que combina la definición de reglas sobre datos de fuentes externas al propio proceso, y la monitorización y validación durante la ejecución de los mismos. Junto a los mecanismos necesarios para la validación de datos, se realiza una diagnosis de datos y reglas de cumplimiento utilizando técnicas de Constraint Programming. Esta metodología puede ser llevada a cabo incluso antes de que todos los datos involucrados en una instancia de proceso hayan sido instanciados, combinando distintos grados de incertidumbre. La propuesta se completa con una gramática para la descripción de las reglas de datos en procesos que permite a los expertos de negocio utilizar lenguajes más cercanos al negocio. El conjunto de soluciones descritas en este artículo han sido validadas sobre un caso real, y ha dado como resultado el desarrollo de un prototipo donde se combinan diferentes tecnologías relacionadas con gestores de procesos de negocio, y resolutores de problemas con restricciones.

Ingeniería del Software Guiada por Búsqueda

Ingeniería del Software Guiada por Búsqueda

Coordinadores: José Raúl Romero Salguero y José Francisco Chicano García

Pedro Delgado-Pérez, Inmaculada Medina-Bulo, Sergio Segura, Antonio García-Dominguez y Juan José Domínguez-Jiménez. *Prueba de Mutación Evolutiva Aplicada a Sistemas Orientados a Objetos.* (Completo)

Isabel María Del Águila, José Del Sagrado y Alfonso Bosch. *Flujo de trabajo para la experimentación colaborativa en Ingeniería del Software guiada por búsqueda.* (Completo)

Francisco Palomo-Lozano, Isabel María Del Águila y Inmaculada Medina-Bulo. *Un algoritmo híbrido para el problema NRP con interdependencias.* (Completo)

Francisco Chicano, Miguel Ángel Domínguez, Isabel Del Águila, José Del Sagrado y Enrique Alba. *Dos estrategias de búsqueda anytime basadas en programación lineal entera para resolver el problema de selección de requisitos.* (Completo)

Javier Ferrer, Francisco Chicano, Roberto Erick Lopez-Herrejon y Enrique Alba. *Aplicando programación lineal entera a la búsqueda de conjuntos de productos de prueba priorizados para líneas de productos software.* (Completo)

Aurora Ramírez, José Antonio Molina, José Raúl Romero y Sebastián Ventura. *Estudio de mecanismos de hibridación para el descubrimiento evolutivo de arquitecturas.* (Completo)

Irene Barba, Andreas Lanz, Andrés Jiménez Ramírez, Barbara Weber, Manfred Reichert y Carmelo Del Valle. *Providing Support for the Optimized Management of Declarative Processes.* (Corto)

José Antonio Parejo Maestre, Aurora Ramírez, José Raúl Romero, Sergio Segura y Antonio Ruiz-Cortés. *Configuración guiada por búsqueda de aplicaciones basadas en microservicios en la nube.* (Corto)

Francisco Palomo-Lozano, Antonia Estero-Botaro y Inmaculada Medina-Bulo. *Minimización de conjuntos de casos de prueba en la prueba de mutaciones de composiciones BPEL.* (Corto)

Andrés Jiménez Ramírez, Barbara Weber, Irene Barba y Carmelo Del Valle: *Generating optimized configurable business process models in scenarios subject to uncertainty*. Information and Software Technology 57, 571–594 (DOI:10.1016/j.infsof.2014.06.006). (Relevante)

Aurora Ramírez, José Raúl Romero, Sebastián Ventura: *An approach for the evolutionary discovery of software architectures*. Inf. Sci. 305: 234-255 (2015). (Relevante)

Gustavo G. Pascual, Mónica Pinto, Lidia Fuentes: *Self-adaptation of mobile systems driven by the Common Variability Language*. Future Generation Comp. Syst. 47: 127-144 (2015). (Relevante)

Prueba de Mutación Evolutiva Aplicada a Sistemas Orientados a Objetos

Pedro Delgado-Pérez[1], Inmaculada Medina-Bulo[1], Sergio Segura[2], Antonio García-Domínguez[3] y Juan José Domínguez-Jiménez[1]

[1] Departamento de Ingeniería Informática, Universidad de Cádiz, España
{pedro.delgado, inmaculada.medina, juanjose.dominguez}@uca.es
[2] Departamento de Ingeniería Informática, Universidad de Sevilla, España
sergiosegura@us.es
[3] Department of Computer Science, University of York, United Kingdom
antonio.garcia-dominguez@york.ac.uk

Resumen A pesar del beneficio que puede reportar la prueba de mutaciones en el proceso de prueba de software, el coste que supone su aplicación siempre ha sido visto como un obstáculo para una mayor acogida por parte de la industria. Por esta razón, se han desarrollado diversas técnicas que tratan de paliar el problema, principalmente mediante la reducción del número de mutantes que son generados. Entre ellas se encuentra la Prueba de Mutación Evolutiva (PME), que propone el empleo de algoritmos evolutivos para encontrar un subconjunto de mutantes que presenta mayor posibilidad de ayudar a refinar el conjunto de casos de prueba empleado. La técnica solo había sido probada con éxito en operadores para el lenguaje de programación WS-BPEL. En este artículo se presentan los experimentos llevados a cabo aplicando la técnica de PME con mutantes generados por operadores de mutación para C++ relacionados con la orientación a objetos. Los resultados obtenidos, usando los parámetros considerados como más apropiados para la configuración del algoritmo, revelan que la técnica también es más efectiva que una estrategia aleatoria con operadores de clase para sistemas en C++.
Palabras clave: Prueba de software, prueba de mutaciones, algoritmos evolutivos, C++, orientación a objetos.

1. Introducción

La *prueba de mutaciones* es una técnica de prueba de software que se emplea tanto para evaluar como para mejorar la habilidad que presenta un conjunto de casos de prueba detectando fallos en el código [22]. La técnica se basa en crear versiones modificadas del programa original, que se conocen como *mutantes*. Cada mutante contiene un simple cambio sintáctico que es insertado en el código mediante los *operadores de mutación* definidos para cada lenguaje de programación. Una vez generados los mutantes, estos son ejecutados contra el conjunto de casos de prueba, al igual que el programa original. Como resultado de este proceso, podemos obtener mutantes que han sido *matados* porque la salida para alguno de los casos de prueba difería de la producida por el programa original, o,

por el contrario, mutantes que permanecen *vivos* porque no se observa ninguna diferencia. Estos últimos pueden ayudarnos a generar nuevos casos de prueba para conseguir detectar esas mutaciones, pudiendo también ocurrir que algunas de las mutaciones no hayan en realidad cambiado la semántica del programa y los mutantes generados sean, por tanto, *equivalentes* al programa original.

Debido a la gran cantidad de mutantes que se pueden derivar de un programa, la prueba de mutaciones es conocida por ser una técnica de alto coste computacional. Este problema unido a la ardua tarea de determinar qué mutantes de los que permanecen vivos son equivalentes suponen una traba para su aplicación. Para reducir el elevado coste que presenta la prueba de mutaciones, se han propuesto diferentes técnicas a los largo de los años, principalmente para reducir el número de mutantes a generar. Entre estas técnicas cabe destacar la mutación selectiva [21], la mutación por muestreo [1], la mutación por agrupamiento [11] o la mutación de orden superior [13]. Una novedosa técnica para la reducción del número de mutantes a generar pero manteniendo un alto potencial para refinar el conjunto de casos de prueba es la conocida como *Prueba de Mutación Evolutiva* [9] (PME de ahora en adelante). Esta técnica se basa en el uso de un algoritmo genético para guiar la búsqueda hacia la obtención de un subconjunto de mutantes útiles para la mejora del conjunto de casos de prueba.

Este artículo tiene como objetivo mostrar los resultados que ofrece la PME en sistemas orientados a objetos en lenguaje C++. La técnica había sido empleada anteriormente con composiciones WS-BPEL con buenos resultados en comparación con el uso de una selección puramente aleatoria. Sin embargo, no se habían llevado a cabo estudios posteriores sobre la aplicabilidad de la PME en diferentes contextos. Los resultados experimentales sobre tres programas reales de diversos tamaños muestran que la técnica obtiene buenos resultados con operadores orientados a objetos para C++, validando su uso en diferentes ámbitos.

El otro objetivo principal de este artículo es presentar la herramienta desarrollada para unir el sistema de mutaciones en C++ [3] con el algoritmo genético implementado en la herramienta *GAmera* [8]. Asimismo, se explican los cambios que han sido llevados a cabo para poner en práctica la técnica debido a las particularidades del lenguaje y los operadores de mutación estudiados.

El resto del artículo se estructura de la siguiente manera. En la siguiente sección se comentan los antecedentes y el trabajo relacionado. En la Sección 3 se muestra el funcionamiento del algoritmo genético y se presentan las peculiaridades de la herramienta desarrollada que conecta el algoritmo genético con el sistema de mutaciones en C++. En la Sección 4 se muestran y discuten los resultados obtenidos en el experimento llevado a cabo. En la última sección, se exponen las conclusiones y el trabajo futuro a realizar.

2. Antecedentes y Trabajo Relacionado

2.1. Prueba de Mutación Evolutiva

La *PME* [9] es una de las técnicas más recientemente presentadas de reducción de mutantes y consecuente reducción del coste computacional. El objetivo

de esta técnica es la de generar tan solo un subconjunto del total de mutantes que contenga un gran porcentaje de los mutantes que realmente permiten refinar el conjunto de casos de prueba utilizado, de manera que no se incurra en una pérdida significativa de la efectividad de la prueba de mutaciones. La obtención de mutantes con capacidad de mejorar el conjunto de pruebas se lleva a cabo a través de un algoritmo genético, el cual, mediante su *función de aptitud*, favorece a los mutantes que los autores de la técnica denominan como *mutantes fuertes*, que pueden ser de dos tipos a su vez:

- **Mutantes potencialmente equivalentes:** Son aquellos que, con el actual conjunto de casos de prueba, permanecen vivos. Estos mutantes pueden o bien sugerir la creación de nuevos casos de prueba para matarlos, o bien pueden resultar finalmente en mutantes equivalentes (condición que solo se conoce tras la inspección de los mismos).
- **Mutantes difíciles de matar:** Son aquellos mutantes que son matados por un único caso de prueba que, además, solo mata a ese mutante.

La PME fue empleada para reducir el coste de la prueba de mutaciones en composiciones WS-BPEL. Para ello, se desarrolló la herramienta *GAmera* [8], que implementaba el algoritmo genético propuesto por los autores. Como resultado del uso de GAmera en las tres composiciones analizadas para alcanzar un porcentaje de mutantes fuertes, se obtuvo en todas ellas una reducción mayor del número de mutantes que si los mutantes fuesen elegidos de manera aleatoria. También se demostró que el impacto del uso del algoritmo en el tiempo global era marginal. Por último, se establecieron ciertos parámetros de configuración para optimizar los resultados del algoritmo genético.

2.2. Prueba de Mutaciones para C++

La prueba de mutaciones ha sido aplicada desde sus inicios a una gran variedad de lenguajes de programación, tales como Java [15], SQL [20] o más recientemente Python [6]. Gran parte de las herramientas desarrolladas y lenguajes de programación a los que se ha aplicado están recogidos en un trabajo de Jia y Harman [12]. Sin embargo, hasta hace unos años no había sido prácticamente abordada en relación a C++. Respecto a este lenguaje, un conjunto de operadores de mutación a nivel de clase fueron definidos en cuanto a características del paradigma de orientación a objetos como la herencia, el polimorfismo y la sobrecarga de métodos [4,5]. Posteriormente se establecieron unas pautas para la correcta generación de mutantes a la hora de implementar los operadores de mutación [2]. Finalmente, se presentó la herramienta que aplicaba un subconjunto de los operadores de clase definidos para este lenguaje [3]. Por su parte, Kusano y Wang [14] también desarrollaron una herramienta de mutaciones enfocada a la concurrencia en aplicaciones multihilo en C++.

El sistema de mutaciones para C++ desarrollado por los autores de este trabajo admite las mismas órdenes de ejecución que MuBPEL para composiciones WS-BPEL [10]. MuBPEL es la herramienta subyacente empleada por GAmera

para el análisis, generación y ejecución de mutantes. El hecho de usar la misma arquitectura permite que podamos reutilizar con mayor facilidad el algoritmo genético implementado en GAmera para aplicarlo a ficheros de código C++.

3. Funcionamiento de la Prueba de Mutación Evolutiva

3.1. Algoritmo Genético

GAmera [8] identifica a un mutante de forma unívoca mediante tres campos (ver Figura 1):

- **Operador**: Código representativo del operador que genera el mutante.
- **Localización**: Orden en el cual el operador de mutación inserta las diferentes mutaciones dentro del código analizado.
- **Atributo**: Orden en el cual el operador inserta las diferentes mutaciones en una misma localización.

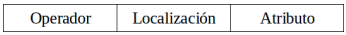


Figura 1. Codificación de un mutante

El algoritmo genético implementado produce una primera generación de individuos (mutantes) aleatoriamente. El tamaño de la población en cada generación viene dado por un parámetro a establecer al inicio. A partir de ahí, produce sucesivas generaciones en las que los nuevos individuos provienen de dos vertientes:

1. **Individuos generados aleatoriamente:** Tal y como sucede en la primera generación, se produce un porcentaje de individuos aleatorios que viene marcado por un parámetro a configurar antes de la ejecución.
2. **Individuos producidos por mutación o por cruce:** El resto de individuos de la generación se producen por esta rama. Se seleccionan de la anterior generación los individuos necesarios mediante el sistema de selección por ruleta (proporcionalmente a la función de aptitud del individuo):
   - **Mutación:** Se muta uno de los campos (operador, localización o atributo) del mutante actual para generar uno nuevo. La mutación del campo realizada se limita para producir un mutante que pueda ser generado. Por ejemplo, si un operador genera mutantes en tres localizaciones, no es posible mutar la localización a cuatro.
   - **Cruce:** Se seleccionan dos individuos que actúan como padres y se selecciona un punto de cruce. Si el punto de cruce se establece entre los campos operador-localización, los padres intercambian el operador manteniendo los campos localización-atributo. Si, por el contrario, el punto de cruce está entre localización-atributo, los padres intercambian el atributo (ver Figura 2).

La probabilidad con la que la herramienta utiliza mutación o cruce también es configurable. Más adelante, en la Sección 3.2, veremos que será necesario modificar estos operadores.

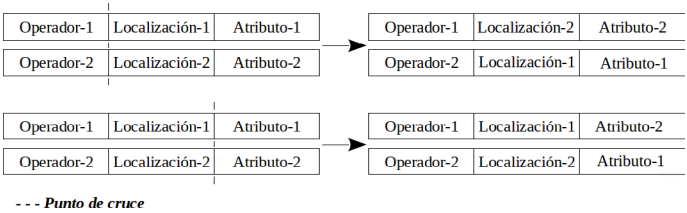


Figura 2. Cruce de mutantes

La función de aptitud para un mutante cuenta el número de casos de prueba que lo matan, así como el número de mutantes que esos casos de prueba matan a su vez. Cuanto mayor sea la suma total de ambas cuentas, peor será la función de aptitud asociada al mutante, pues los casos de prueba que se pueden llegar a generar con ese mutante resultan poco específicos. De esta manera, respecto a los mutantes fuertes (ver Sección 2.1):

- Los mutantes potencialmente equivalentes reciben el valor más alto al no ser matados por ningún caso de prueba.
- Los mutantes difíciles de matar reciben el segundo mejor valor, al ser matados por un solo caso de prueba.

Por el contrario, los mutantes que son matados por todos los casos de prueba son los que reciben las valoraciones más bajas de la función de aptitud. Además de lo dicho, es importante destacar las siguientes características del algoritmo:

- **Uso de segunda población:** El algoritmo utiliza los mutantes generados hasta el momento para estimar de manera más precisa el valor de la aptitud de cada individuo, de manera que la aptitud de un mutante puede variar de una generación a otra.
- **Normalización de valores:** Dado que cada operador genera un número de mutantes distintos en el sistema, para que todos los mutantes tengan la misma probabilidad de ser seleccionados, el sistema normaliza/desnormaliza los valores de los campos localización y atributo.

3.2. Prueba de Mutación Evolutiva para Sistemas Orientados a Objetos en C++

La prueba de mutaciones es una técnica de caja blanca que requiere de marcos de trabajo particulares para cada lenguaje de programación. En este sentido, se

ha desarrollado una herramienta que permite aplicar la PME a aplicaciones codificadas en lenguaje C++. El principal cometido de esta herramienta es la de coordinar la acción entre otros dos sistemas existentes:

- **GAmera** [8]: Esta aplicación implementa el algoritmo evolutivo propuesto por los creadores de la técnica [9].
- **Sistema de mutaciones para C++** [3]: Esta herramienta aplica la prueba de mutaciones al lenguaje C++ mediante operadores a nivel de clase [5].

GAmera fue utilizada para la obtención de resultados experimentales con composiciones en lenguaje WS-BPEL. No obstante, la herramienta está modularizada de manera que el algoritmo genético es independiente del lenguaje al que se quiera aplicar. De esta manera, la herramienta que se ha desarrollado hace corresponder los datos entre GAmera y el sistema de mutaciones para que la conexión entre estos dos sistemas sea posible. La nueva herramienta lleva a cabo las siguientes acciones:

- Transformar las órdenes de ejecución que GAmera requiere para su funcionamiento (análisis, generación y ejecución de mutantes) en órdenes entendibles para el sistema de mutaciones en C++.
- Traducir la salida generada por el sistema de mutaciones al formato de entrada para GAmera y, al contrario, formatear la salida de GAmera como entrada del sistema de mutaciones.

Debido a las características de los dos sistemas a los que esta herramienta conecta, ha sido necesario llevar a cabo varias modificaciones respecto al planteamiento inicial de la técnica. Estos cambios, sin embargo, pueden afectar al funcionamiento del algoritmo genético, tal y como se explica a continuación:

1. Como se comentó en la sección anterior, GAmera identifica a un mutante mediante tres campos: operador, localización y atributo. El campo atributo depende de cada operador de mutación concreto. En este sentido, podemos encontrar las siguientes opciones:
   - **Atributo fijo**: El número de posibles mutaciones a insertar es conocido de antemano. Por ejemplo, un operador que reemplaza un operador aritmético puede reemplazar el operador por un conjunto de operadores aritméticos conocido.
   - **Atributo variable**: El número de posibles mutaciones depende por completo de la localización exacta dentro del código. A modo de ejemplo, un operador de mutación que reemplaza una llamada a un método por otro método que sea compatible (a fin de que no genere un mutante inválido) dependerá del número de métodos compatibles que existan.

   Cuando el algoritmo genético elige el campo "atributo" de un individuo para ser mutado, es necesario que el atributo sea fijo para que el algoritmo pueda seleccionar con igualdad de posibilidades el campo atributo que tendrá el nuevo individuo generado. Es por ello que, cuando un operador de mutación presenta un atributo variable, el atributo se marca siempre como "1" y las diferentes variaciones que se pueden insertar en esa localización se contabilizan simplemente como nuevas localizaciones.

Dicho esto, los operadores de clase implementados en el sistema de mutaciones para C++ o bien solo tienen la posibilidad de generar un único mutante por localización o bien su atributo es variable. Esto quiere decir que, en caso de que el algoritmo genético elija el campo atributo para ser mutado y generar un nuevo individuo, como el campo atributo siempre tiene valor "1", nunca se generaría un individuo diferente a partir de la mutación de ese campo. Es por ello por lo que, para realizar los experimentos, se ha añadido una nueva opción a GAmera para permitir la posibilidad de que el campo atributo no sea mutado para una ejecución dada. Esto también se aplica a la mutación por cruce que intercambia el atributo.

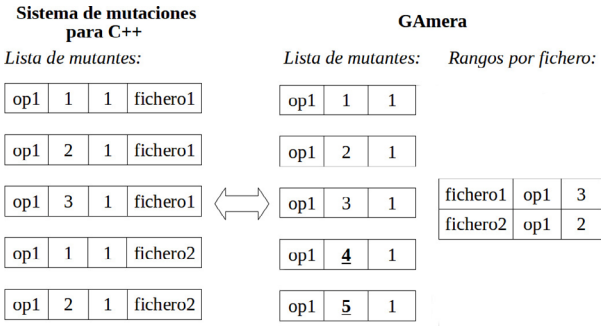


Figura 3. Ejemplo de traducción de mutantes entre el sistema de mutaciones para C++ y GAmera de un operador "op1" aplicado a dos ficheros "fichero1" y "fichero2".

2. Las composiciones WS-BPEL que recibe la herramienta MuBPEL [10] están codificadas en un único fichero fuente. Sin embargo, las aplicaciones en C++ suelen componerse de varios ficheros. El sistema de mutaciones para C++ permite la posibilidad de mutar más de un fichero en la misma ejecución. Para ello, la herramienta añade un campo adicional a la 3-tupla (operador, localización, atributo) para identificar unívocamente cada mutante: el *nombre del fichero* al que pertenece. GAmera solo está configurado para tratar los campos (operador, localización, atributo), de manera que ha sido necesario realizar una traducción entre la salida del sistema de mutaciones y la entrada de GAmera. Esta traducción consiste en hacer corresponder los mutantes entre ambas herramientas mediante la codificación/decodificación del nombre del fichero a través de la localización. Este proceso se detalla gráficamente en la Figura 3.

   No obstante, el hecho de permitir que más de un fichero de código sea mutado en la misma ejecución produce un efecto en el algoritmo genético no

contemplado en los experimentos originales con respecto a WS-BPEL. Tanto la mutación del campo "operador" como del campo "localización" puede provocar que el nuevo individuo generado corresponda a una clase que esté en un fichero de código diferente al del individuo que fue mutado. Aunque las clases de un mismo programa suelen usar un mismo patrón de diseño y estar interrelacionadas, el comportamiento de un operador en clases diferentes puede variar, principalmente cuando se encuentran en ficheros de código distintos. Por tanto, este hecho puede afectar al algoritmo genético puntualmente, pero también puede suponer una ventaja al permitir al algoritmo abrirse hacia nuevos espacios de búsqueda.

4. Experimentos

4.1. Casos de estudio

Para los experimentos conducidos en este artículo, se ha hecho uso de tres programas reales de código abierto:

1. **Dolphin** [7]: Administrador de archivos en aplicaciones de escritorio de KDE por defecto.
2. **Tinyxml2** [19]: Analizador sencillo y eficiente de código XML.
3. **QTDom** [16]: Módulo de Qt que ofrece una implementación en C++ del estándar DOM.

En cada aplicación, los mutantes que se pueden generar con los operadores de mutación implementados en el sistema de mutaciones son siempre los mismos, es decir, los mutantes no varían en diferentes ejecuciones. Por lo tanto, a partir del resultado de una ejecución previa de todos los mutantes en cada aplicación, es posible conocer el número total de mutantes fuertes existentes. Tal y como puede verse en el Cuadro 1, estas aplicaciones producen un número diferente de mutantes totales así como de mutantes fuertes, lo cual nos permite observar el funcionamiento de la técnica en relación a la cantidad de mutantes que se genera. Además, en esta tabla también puede observarse el tamaño del conjunto de casos de prueba empleado para cada uno de los programas, el cual es el que originalmente viene distribuido con cada aplicación.

Cuadro 1. Mutantes y conjunto de casos de prueba de las aplicaciones del estudio

	Dolphin	**Tinyxml2**	**QtDom**	**Total**
Mutantes totales	219	614	1,146	1,979
Válidos	208	433	681	1,322
Fuertes	103	159	348	610
% Mutantes fuertes	49.5%	36.7%	51.1%	46.1%
Casos de prueba	61	57	46	164

Cuadro 2. Parámetros utilizados en la ejecución del algoritmo evolutivo

Parámetro	**Valor**
Tamaño población	5%
Individuos generados aleatoriamente	10%
Individuos generados por mutación o cruce	90%
- **Probalididad de mutación**	30%
- **Probabilidad de cruce**	70%

4.2. Configuración del Experimento

El experimento llevado a cabo busca analizar el resultado obtenido por la PME en comparación con una técnica aleatoria:

1. **Aleatoria:** Consiste en ordenar aleatoriamente los mutantes al inicio y después ir seleccionándolos de uno en uno hasta llegar a la condición de parada.
2. **PME:** A partir de la población inicial, se irán produciendo nuevas generaciones de mutantes en base a los parámetros establecidos para el funcionamiento del algoritmo evolutivo. Los parámetros empleados son los que se pueden observar en el Cuadro 2, los cuales son los que en los experimentos realizados en el artículo en el que se presenta la técnica fueron hallados como los más adecuados [9]. El tamaño de la población se refiere al número de mutantes que serán producidos en cada generación, el cual es un porcentaje respecto del total de mutantes en cada aplicación. Como puede observarse, la suma del porcentaje de mutantes generados aleatoriamente y por mutación/cruce es de 100 %, pues a través de estas dos vertientes se crean todos los mutantes de una generación.

En este experimento se busca localizar a los mutantes fuertes, por lo cual se han establecido dos condiciones de parada: al llegar al 75 % y al 90 % del total de mutantes fuertes. Como se mencionó anteriormente, este criterio es posible establecerlo ya que, debido a una ejecución previa, conocemos el número de mutantes fuertes existentes. La herramienta ha sido configurada para medir tanto el porcentaje de mutantes generados hasta el momento de llegar a los límites de parada como el tiempo total necesario para la ejecución de los casos de prueba en estos mutantes. Para evitar el sesgo que puede producirse en una única ejecución de las técnicas (debido al componente aleatorio que ambas conllevan), se han realizado 30 ejecuciones con diferentes semillas de partida. De esta manera, los cálculos a mostrar se obtienen a partir de los datos de estas 30 ejecuciones.

Es necesario remarcar que, para saber si un mutante es o no fuerte, es necesaria la ejecución de todos los casos de prueba (por lo que no basta parar la ejecución al primer caso de prueba que mata al mutante). Asimismo, el algoritmo evolutivo utilizará esta información para el cálculo de la función de aptitud.

4.3. Resultados

Los resultados del experimento se recogen en el Cuadro 3 y 4. En el Cuadro 3 se muestra la media, mediana, valores mínimos y máximo y desviación estándar

en cuanto al porcentaje de mutantes que se necesita generar para obtener un 75 y un 90 % de los mutantes considerados como fuertes. En esta tabla se presentan los datos tanto de la PME como de la técnica aleatoria para los tres casos de estudio. Como puede observarse, el uso del algoritmo genético nos conduce a producir un porcentaje menor de mutantes en todos los casos para lograr los porcentajes de mutantes fuertes establecidos (75 % y 90 %). Especialmente destacable es la diferencia en promedio en torno al 10 % conseguida respecto de la técnica aleatoria en *Tinyxml2*. También podemos observar que la diferencia entre ambas técnicas se reduce al llegar a un umbral más alto (en este caso del 90 %). Por ejemplo, refiriéndonos nuevamente a *Tinyxml2*, la diferencia se acorta en algo más de 4 puntos porcentuales.

En el Cuadro 4 se indica el tiempo en minutos que tardó la ejecución total de cada caso de prueba contra cada mutante generado antes de llegar a la condición de parada. Al igual que en el cuadro anterior, se calcula la media, mediana, mínimo, máximo y desviación estándar. Hay que hacer notar en este punto que los tiempos mostrados para cada una de las aplicaciones varían debido a que cada una de ellas es estudiada junto con un conjunto de casos de prueba, cada uno de los cuales toma un tiempo distinto en ejecutarse. El tiempo que tarda la PME es menor en todos los casos en los distintos parámetros estadísticos mostrados. No obstante, excepto en *Tinyxml2* y límite *75 %*, la desviación estándar es menor usando aleatoriedad que la técnica evolutiva.

4.4. Análisis y discusión de resultados

La PME ha mostrado necesitar generar menos mutantes para conseguir producir una mayor cantidad de mutantes fuertes que la selección aleatoria en los tres casos de estudio analizados. Cuando buscamos un 75 % de los mutantes fuertes, en promedio el ahorro en cuanto al número de mutantes es de alrededor al 6.6 %, siendo destacable el caso de *Tinyxml2* en más de un 10 %. El resultado sigue siendo mejor que la técnica aleatoria cuando se trata de encontrar un porcentaje muy elevado de mutantes fuertes (90 %), aunque la mejora en promedio baja al 3.7 %. Esto puede ser debido en parte a la existencia de unos pocos mutantes fuertes en operadores que en su mayoría generan mutantes considerados de baja calidad para el algoritmo genético. A fin de conocer si estas diferencias son estadísticamente significativas (p-*value*<0.01), también se ha realizado un estudio estadístico con la aplicación *STATService* [18], que aplica el test apropiado para los datos de las distintas ejecuciones (*Test* en Cuadro 3). En concreto, los test estadísticos aplicados han sido *Wilcoxon* para el caso *Tinyxml2* con umbral 90 % y *T de student* para los otros 5 casos. Los resultados en todos los casos nos llevan a aceptar que la mediana de los porcentajes de mutantes generados por la técnica evolutiva para generar un porcentaje de mutantes fuertes es significativamente menor que la aleatoria. Como es de esperar, el tiempo es también mejor en todos los casos ya que el número de mutantes a ejecutar es siempre menor. El ahorro promedio del tiempo es de 3.3 y 2.7 minutos en la ejecución de los casos de prueba para el umbral del 75 % y del 90 % respectivamente.

Cuadro 3. Porcentaje de mutantes generados por la técnica de PME y la técnica aleatoria hasta conseguir un 75 % y 90 % de los mutantes fuertes.

Umbral	***75%***		***90%***	
Técnica	*PME*	*Aleatoria*	*PME*	*Aleatoria*
Dolphin				
Media	69.87	75.11	85.35	89.07
Mediana	70.09	75.79	85.38	89.72
Mínimo	62.55	67.57	78.99	82.64
Máximo	76.71	81.27	90.41	93.15
D.E.	3.57	3.57	2.67	2.86
Test	p-value	4.55×10^{-07}	p-value	2.72×10^{-06}
Tinyxml2				
Media	64.91	74.93	84.32	89.98
Mediana	64.74	74.83	84.12	90.22
Mínimo	60.58	69.70	77.85	85.01
Máximo	71.49	80.61	89.73	93.64
D.E.	2.59	2.78	3.34	1.78
Test	p-value	7.14×10^{-21}	p-value	1.65×10^{-06}
QtDom				
Media	69.96	74.43	87.84	89.76
Mediana	70.15	74.34	88.09	89.75
Mínimo	66.05	71.64	83.33	86.21
Máximo	73.38	78.88	90.13	93.71
D.E.	1.98	2.00	1.60	1.58
Test	p-value	4.31×10^{-12}	p-value	1.71×10^{-05}

A la vista de los resultados, parece que no hay una gran dependencia del algoritmo genético en relación a la cantidad de mutantes producida en cada caso de estudio. A diferencia de los resultados de los experimentos en [9], el porcentaje más bajo de mejora se obtiene en el caso de estudio que engendra más mutantes. Por lo tanto, de los resultados de este experimento no podemos concluir que la técnica escale en proporción al tamaño del programa analizado, tal y como lo hacen los autores de la técnica en su artículo.

No obstante, analizando los resultados del Cuadro 1, parece que el beneficio reportado por el algoritmo genético está más en relación a la cantidad de mutantes fuertes de partida existentes. Como puede observarse en la tabla, *Tinyxml2* es el que tiene un menor porcentaje de mutantes fuertes (36.8 %), seguido de *Dolphin* (49.5 %) y *QtDom* (51.1 %). De hecho, volviendo a los resultados del Cuadro 3, el porcentaje de mutantes a generar por la técnica evolutiva en promedio también está en consonancia con el orden respecto al porcentaje de mutantes fuertes en estos programas. Por ejemplo, mirando el umbral del 75 %, la técnica evolutiva genera el 64.91 % del total de mutantes para *Tinyxml2*, mientras que requiere de la creación del 69.87 % y 69.96 % de los mutantes para *Dolphin* y *QtDom* respectivamente. Los operadores de mutación a nivel de clase son conocidos

Cuadro 4. Tiempo de ejecución de los mutantes generados por la técnica de PME y la técnica aleatoria hasta conseguir un 75 % y 90 % de los mutantes fuertes.

Umbral	***75%***		***90%***	
Técnica	*PME*	*Aleatoria*	*PME*	*Aleatoria*
Dolphin				
Media	23.78	27.57	29.96	32.39
Mediana	23.54	28.16	29.77	33.15
Mínimo	18.06	19.76	24.81	26.65
Máximo	29.86	31.10	33.75	35.00
D.E.	3.24	2.62	2.36	2.28
Tinyxml2				
Media	19.11	23.61	20.45	24.54
Mediana	18.95	23.60	20.56	24.60
Mínimo	17.08	21.52	18.22	22.80
Máximo	21.56	25.16	21.99	25.34
D.E.	0.96	0.98	1.08	0.59
QtDom				
Media	7.68	9.43	7.84	9.43
Mediana	7.69	9.47	7.82	9.38
Mínimo	6.94	8.71	7.15	8.91
Máximo	8.47	9.89	8.78	9.91
D.E.	0.38	0.29	0.39	0.28

por producir un porcentaje de mutantes equivalentes superior a los operadores tradicionales [17], pero aun así los resultados al comparar ambas técnicas se han mostrado positivos a favor del algoritmo evolutivo también en estos casos.

Por todo lo anterior, concluimos que la PME es una técnica beneficiosa ya que con ella seremos capaces de producir un porcentaje menor de mutantes (consecuentemente, mejorando el coste computacional), aun así conteniendo un elevado porcentaje de los mutantes que nos ayudan a refinar nuestro conjunto de casos de prueba. Los resultados evidencian que cuanto menor es el porcentaje de mutantes fuertes, lo cual suele ocurrir cuando el conjunto de casos de prueba posee ya una capacidad significativa de detección de fallos, mejor serán los resultados de la aplicación de la técnica.

5. Conclusiones y Trabajo Futuro

En este trabajo se presenta una evaluación de la técnica conocida como PME con operadores a nivel de clase para el lenguaje C++. Para ello ha sido necesario desarrollar una aplicación que conectara la herramienta que implementa el algoritmo genético y la que genera los mutantes para C++, introduciendo cambios específicos ajustados a las necesidades del lenguaje y operadores a analizar. Esta técnica de reducción de mutantes, que solo había sido aplicada hasta el momento a composiciones WS-BPEL, ha mostrado ofrecer mejores resultados que la

selección aleatoria de mutantes a la hora de encontrar aquellos mutantes que pueden ayudarnos a mejorar nuestro conjunto de casos de prueba, posibilitando el objetivo de la técnica que es la reducción del coste computacional. El hecho de que la técnica haya demostrado su utilidad para lenguajes y operadores de mutación diferentes, convierte a la PME en una técnica a tener en cuenta para el proceso de mejora de los casos de prueba en cualquier contexto. Como dato novedoso, a la luz de los resultados parece que el rendimiento de la PME respecto a la técnica aleatoria mejora cuanto menor es el porcentaje de mutantes fuertes existentes, y no depende significativamente del número de mutantes total, tal y como indicaban los primeros experimentos en los que se aplicaba la técnica. A pesar de que usando operadores a nivel de clase es probable encontrar un porcentaje elevado de mutantes potencialmente equivalentes, la técnica ha obtenido mejores resultados también en estos casos en nuestros experimentos. No obstante, en nuestros experimentos se han usado tres casos de estudio y sería conveniente confirmar esta tendencia en nuevos experimentos con un número mayor de aplicaciones.

Como trabajo futuro, resultaría interesante repetir estos experimentos para además de calcular el porcentaje de mutantes fuertes, comprobar en qué medida esta técnica nos ayuda realmente a refinar un conjunto de casos de prueba antes que otras técnicas. Asimismo, se podrían aplicar cambios al algoritmo genético para introducir nuevas variables, como la introducción de un porcentaje de los mutantes generados en una generación en la siguiente.

Agradecimientos: Este trabajo fue parcialmente financiado por la beca de investigación PU-EPIF-FPI-PPI-BC 2012-037 de la Universidad de Cádiz, por la Red de Excelencia SEBASENET (TIN2015-71841-REDT), por los proyectos nacionales del Ministerio de Economía y Competitividad DArDOS (TIN2015-65845-C3-3-R) y TAPAS (TIN2012-32273), y por los proyectos de la Junta de Andalucía THEOS (TIC-5906) y COPAS (P12-TIC-1867).

Referencias

1. Budd, T.A.: Mutation Analysis of Program Test Data. Ph.D. thesis, Yale University (1980)
2. Delgado-Pérez, P., Medina-Bulo, I., Domínguez-Jiménez, J.J.: Generación de mutantes válidos en el lenguaje de programación C++. In: XIX Jornadas de Ingeniería del Software y Bases de Datos, JISBD 2014. Cádiz, Spain (2014)
3. Delgado-Pérez, P., Medina-Bulo, I., Domínguez-Jiménez, J.J.: Herramienta para la prueba de mutaciones en el lenguaje C++. In: XX Jornadas de Ingeniería del Software y Base de Datos, JISBD 2015. Santander, Spain (2015)
4. Delgado-Pérez, P., Medina-Bulo, I., Domínguez-Jiménez, J.J., García-Domínguez, A.: Operadores de mutación a nivel de clase para el lenguaje C++. In: XII Jornadas sobre Programación y Lenguajes, PROLE 2013. Madrid, Spain (2013)
5. Delgado-Pérez, P., Medina-Bulo, I., Domínguez-Jiménez, J.J., García-Domínguez, A., Palomo-Lozano, F.: Class mutation operators for C++ object-oriented systems. Annals of telecommunications 70(3-4), 137–148 (2015), http://dx.doi.org/10.1007/s12243-014-0445-4

6. Derezinska, A., Halas, K.: Experimental evaluation of mutation testing approaches to Python programs. In: Proceedings of the 9th Workshop on Mutation Analysis (MUTATION'14). pp. 156–164. Cleveland, USA (March 2014), http://dx.doi.org/10.1109/ICSTW.2014.24
7. Dolphin. https://www.kde.org/applications/system/dolphin, [Online; accessed 14-March-2016]
8. Domínguez-Jiménez, J.J., Estero-Botaro, A., García-Domínguez, A., Medina-Bulo, I.: GAmera: an automatic mutant generation system for WS-BPEL compositions. In: Eshuis, R., Grefen, P., Papadopoulos, G.A. (eds.) Proceedings of the 7th IEEE European Conference on Web Services. pp. 97–106. IEEE Computer Society Press, Eindhoven, The Netherlands (Nov 2009)
9. Domínguez-Jiménez, J.J., Estero-Botaro, A., García-Domínguez, A., Medina-Bulo, I.: Evolutionary mutation testing. Information and Software Technology 53(10), 1108–1123 (Oct 2011), http://dx.doi.org/10.1016/j.infsof.2011.03.008
10. García-Domínguez, A., Estero-Botaro, A., Medina-Bulo, I., Palomo-Lozano, F.: Herramienta de mutación firme para WS-BPEL 2.0. In: Actas de las XVII Jornadas de Ingeniería del Software y Bases de Datos JISBD 2012. pp. 415–418 (2012)
11. Hussain, S.: Mutation Clustering. Master's thesis, King's College London (2008)
12. Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. Software Engineering, IEEE Transactions on 37(5), 649 –678 (Oct 2011), http://dx.doi.org/10.1109/TSE.2010.62
13. Jia, Y., Harman, M.: Higher order mutation testing. Information and Software Technology 51(10), 1379–1393 (Oct 2009), http://dx.doi.org/10.1016/j.infsof.2009.04.016
14. Kusano, M., Wang, C.: CCmutator: A mutation generator for concurrency constructs in multithreaded C/C++ applications. In: Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on. pp. 722–725. IEEE (2013), http://dx.doi.org/10.1109/ASE.2013.6693142
15. Ma, Y.S., Offutt, J., Kwon, Y.R.: MuJava: An automated class mutation system: Research articles. Software Testing, Verification and Reliability 15(2), 97–133 (Jun 2005), http://dx.doi.org/10.1002/stvr.v15:2
16. QtDOM. https://github.com/qtproject/qtbase/tree/dev/src/xml/dom, [Online; accessed 14-March-2016]
17. Segura, S., Hierons, R.M., Benavides, D., Ruiz-Cortés, A.: Mutation testing on an object-oriented framework: An experience report. Information and Software Technology 53(10), 1124–1136 (2011), http://dx.doi.org/10.1016/j.infsof.2011.03.006, special Section on Mutation Testing
18. STATService. http://moses.us.es/statservice, [Online; accessed 20-June-2016]
19. Tinyxml2. https://github.com/leethomason/tinyxml2, [Online; accessed 14-March-2016]
20. Tuya, J., Suárez-Cabal, M.J., la Riva, C.d.: Mutating database queries. Information and Software Technology 49(4), 398–417 (Apr 2007), http://dx.doi.org/10.1016/j.infsof.2006.06.009
21. Wong, W.E., Mathur, A.P.: Reducing the cost of mutation testing: An empirical study. techreport, Purdue University, West Lafayette, Indiana (1993)
22. Woodward, M.R.: Mutation testing - its origin and evolution. Information and Software Technology 35(3), 163–169 (Mar 1993), http://dx.doi.org/10.1016/0950-5849(93)90053-6

Flujo de trabajo para la experimentación colaborativa en Ingeniería del Software guiada por búsqueda

Isabel María del Águila, José del Sagrado y Alfonso Bosch

Departamento de Informática, Universidad of Almería,
Ctra. de la Playa s/n, 04120 Almería, Spain
{imaguila,jsagrado,abosch}@ual.es

Resumen La Ingeniería del Software Guiada por Búsqueda persigue reformular problemas de Ingeniería del Software que a menudo comprenden objetivos en conflicto, como problemas de optimización. Así, las técnicas que se aplican en esta disciplina buscan una o un conjunto de soluciones casi-óptimas en un espacio de soluciones candidatas con la ayuda de una función de aptitud que les permita distinguir las mejores soluciones. La naturaleza estocástica de los algoritmos de optimización requiere de la repetición de las búsquedas para mitigar los efectos de la aleatoriedad. A la hora de comparar algoritmos, el investigador comparará los resultados con mejor calidad (mejores valores en la función de aptitud, en indicadores de calidad y rendimiento) devueltos en las búsquedas, lo que conlleva un trabajo adicional por parte del investigador. La sobrecarga que implica esta actividad puede aminorarse si la experimentación se enfoca de manera colaborativa. Este artículo propone un flujo de trabajo para la experimentación colaborativa basado en resultados e indicadores de calidad y rendimiento.

Keywords: metaheurísticas, experimentación cooperativa, indicadores de calidad

1. Introducción

La Ingeniería del Software basada o guiada por búsqueda une los campos de algoritmos de Inteligencia Artificial e Ingeniería de Software. En la literatura encontramos numerosos trabajos que ponen en valor esta colaboración y que según Harman et al. [15] se pueden clasificar en tres áreas principales: 'Clasificación, aprendizaje y predicción en Ingeniería del Software', 'Ingeniería del Software probabilística' e 'Ingeniería del Software guiada por búsqueda'. En clasificación, aprendizaje y predicción algunos autores proponen modelos de predicción del riesgo [18], o bien estudian los defectos [17] o predicen el esfuerzo necesario para la ejecución del proyecto [31]. Técnicas probabilísticas basadas en redes Bayesianas modelan cuestiones de Ingeniería del Software para el control de calidad [19], la predicción de defectos [29] o el estudio de los requisitos [4] La Ingeniería

del Software guiada por búsqueda es un campo de investigación cuyo objetivo es reformular problemas de Ingeniería del Software para automatizar la construcción de soluciones a los problemas que abarcan desde el análisis de requisitos hasta la refactorización y mantenimiento [15].

Una vez que un problema de Ingeniería del Software se define como un problema de búsqueda pueden aplicársele diferentes aproximaciones metaheurísticas que nos darán soluciones de distinta calidad, lo que unido a la naturaleza estocástica de los algoritmos de optimización, hace que, en la mayoría de los casos, sea necesario recurrir a la experimentación, en lugar de al análisis puramente teórico, a la hora de evaluar las propuestas de investigación en este campo de investigación.

Un experimento consiste en la manipulación y observación de la realidad para el estudio detallado de un fenómeno. En Ingeniería del Software es difícil describir los experimentos con el detalle suficiente como para que otros puedan reproducirlos dentro de sus propios resultados de trabajo [16][11]. En el campo de las metaheurísticas existen entornos de trabajo que permiten manipular los experimentos y los resultados de estos [22], incluso definiendo lenguajes y entornos para describir con precisión experimentos centrados en optimización con metaheurísticas, (e.g. https://examplar.us.es) Pero, en general, cuando buscamos la utilización del método científico experimental para la generación de nuevo conocimiento sobre el proceso de desarrollo de software, estas aproximaciones deben ser recubiertas con un nivel superior de abstracción, donde se defina un marco unificado de los trabajos partiendo de la Ingeniería del Software, no sólo centrado en los algoritmos y sus soluciones.

En este trabajo definimos un proceso común que ayude a los investigadores a mantener un patrón estándar y que además, permita a los participantes de los ensayos de Ingeniería del Software guiada por búsqueda colaborar para la comparación de los algoritmos y la reproducibilidad de los experimentos, haciendo así verificables los nuevos avances propuestos en cada caso.

El resto del trabajo se organiza como sigue. En la sección 2 se enmarca el trabajo dentro de la experimentación colaborativa en disciplinas científicas. A continuación, la sección 3 describe el flujo de trabajo genérico para los ensayos de Ingeniería del Software guiada por busqueda. Una vez descrito, se instancia para el caso específico del problema de selección de los requisitos de la siguiente versión del software en la sección 4. Para finalizar, en la sección 5 se presentan las conclusiones y se esbozan posibles líneas de trabajo futuro.

2. Experimentación colaborativa

Unificar las características específicas de experimentación en Ingeniería de Software y metaheurísticas no es una tarea trivial, se requieren mecanismos e instrumentos que permitan la comunicación entre investigadores. Por una parte, es necesario mejorar los procedimientos para la investigación en metaheurísticas incrementando la trasparencia de las implementaciones y la compresión de cada una de las partes que las componen [28]. Y por otra, puesto que en ciencia e

ingeniería no tiene sentido hablar de experimentos aislados, el resultado de un solo experimento no puede ser representativo. En la mayoría de los casos la experimentación necesita poder ser validada por replicación o al menos debe existir la posibilidad de comparar, de forma efectiva, los resultados de diversos ensayos utilizando medidas o indicadores de calidad a nivel del algoritmo en sí y del problema al que se busca solución.

De cara a ser capaces de hacer útiles y efectivos los ensayos en el campo de la Ingeniería del Software guiada por búsqueda necesitamos unificar el proceso y así hacer que los resultados sean comparables. La responsabilidad no se debe centrar en un sólo investigador, sino más bien en la comunidad [10], por lo que este proceso unificado debe abordarse desde una perspectiva colaborativa. Lograr la comunicación entre los investigadores es una tarea compleja, que supone inicialmente sobrepasar barreras para la mera transferencia de conocimiento. Es necesario acordar cuáles, cómo y con qué se ejecutarán las distintas etapas de los ensayos experimentales que permitirán estudiar la validez de la aplicación de las técnicas de búsqueda a los problemas específicos de la Ingeniería del Software. Además, cada tipo de técnica presenta sus propias particularidades: no es lo mismo aplicar el recocido simulado que un algoritmo genético o un resolutor basado en programación lineal. Del mismo modo, no es igual tratar con el problema de selección de requisitos que con el de selección de los casos de prueba o de la arquitectura de software. Cada uno presenta dimensiones propias en la toma de decisiones y para definir la mejor solución. Es decir, además de los indicadores de calidad a nivel puramente de la técnica de búsqueda utilizada, se necesita utilizar indicadores de calidad específicos de cada tipo de problema [2]. La comparabilidad por tanto se definirá en torno a indicadores que representen la abstracción y encapsulación del ensayo realizado.

Una nueva aportación en esta disciplina se debe articular en torno a actividades genéricas, con una clara definición de cómo éstas se comunican entre sí. Las tareas individuales pueden desarrollarse por el mismo investigador que propone el trabajo o bien pueden ejecutarse, o haber sido ejecutadas, por otros investigadores. Por ejemplo, si queremos proponer una nueva técnica metaheuristica no sería necesario volver a realizar la implementación de otras técnicas contra las que realizar la comparativa, siempre y cuando el ensayo respete la estructura definida en las tareas genéricas y se pueda trabajar a nivel de indicadores. También será posible colaborar compartiendo las rutinas implementadas con tal de que exista y se cumplan los protocolos de interfaz.

3. Actividades genéricas de colaboración

De forma general, los ensayos experimentales en Ingeniería del Software guiada por búsqueda son costosos de organizar. Si extrapolamos las ventajas generales de la estandarización de metaheurísticas [20], tener una referencia estándar que pueda adaptarse o instanciarse en cada caso sería beneficioso en diversos aspectos: *menor dependencia de los investigadores* puesto que se facilitará la automatización, *mejorar la transparencia* al registrar los procesos y, por tanto,

saber dónde se ha de *tomar cada decisión* sobre el ensayo, *fomentar la reproducibilidad* al explicitar las condiciones experimentales, *facilitar las tareas de análisis de los resultados* definiendo los procesos y medidas estadísticas, así como los indicadores que abstraen los datos en bruto y ,finalmente, *permitir compartir y generar conocimiento*, puesto que tener un marco de referencia permitirá trabajar cooperativamente distribuyendo responsabilidades y sacar partido de la experiencia individual.

Este escenario colaborativo se tiene que abordar desde distintos puntos de vista o niveles. Para eso hemos representado con piscinas o carriles los diversos participantes en el proceso, (figura 1). Por una parte está el responsable del ensayo que representa al equipo de investigación que plantea la definición de una nueva técnica a un problema dado. Por otra está el marco colaborativo, que describe las tres grandes responsabilidades que aparecen cuando se realiza un ensayo. Estas tareas no tienen porqué ser realizadas por los investigadores responsables del ensayo, sino que pueden delegar en trabajos previos o paralelos desarrollados por otros equipos, siempre y cuando se acojan a esta marco de referencia.

Los aspectos básicos de un ensayo de Ingeniería del Software guiada por búsqueda se articulan en tres niveles de responsabilidades:

- *El nivel del conjunto de datos sobre el que realizar los ensayos.* Debido a las políticas de confidencialidad de la mayoría de las empresas de desarrollo de software comercial, disponer de datos sobre los que aplicar los ensayos no es nada fácil. Una alternativa es utilizar datos publicados en los trabajos de investigación [7,33]. Otra es extraer los datos de los repositorios públicos de proyectos de código abierto [32,12]. Pero no existen formatos unificados para la especificación de estos datos de acuerdo con el problema para el que van a ser utilizados. Además, en ocasiones se deben trasformar los datos para adaptarlos al problema, como el caso presentado en Sagrado et al, 2015 [25] donde se muestra como trasformar un problema con dependencias entre requisitos a un conjunto de datos sin ellas.
  Con lo cual, junto con la selección y formulación del problema, se tendrá que construir el conjunto de datos sobre los que ejecutar los experimentos. Se puede recuperar una formulación previa o definir una nueva propuesta lo que implica, por ejemplo, seleccionar entre estrategias monobjetivo o multiobjetivo. El escenario de prueba puede ser cualquiera de los ya unificados para ese problema o bien necesitar la trasformación o extracción de los datos para generar un nuevo escenario, que además generará un nuevo caso disponible para el resto de la comunidad.
- *El nivel de la técnica inteligente a aplicar.* Las metaheurísticas han sido las preferidas para afrontar problemas de Ingeniería del Software guiada por búsqueda, pero se han aplicado con éxito otras técnicas [14,3]. La estandarización y cooperación en la experimentación con metaheuríticas tiene la ventaja de mejorar la comunicación y reproducibilidad, la interoperatividad, el ensamblado automático de las metaheurísticas, la generación de conocimiento y la eficiencia [28].

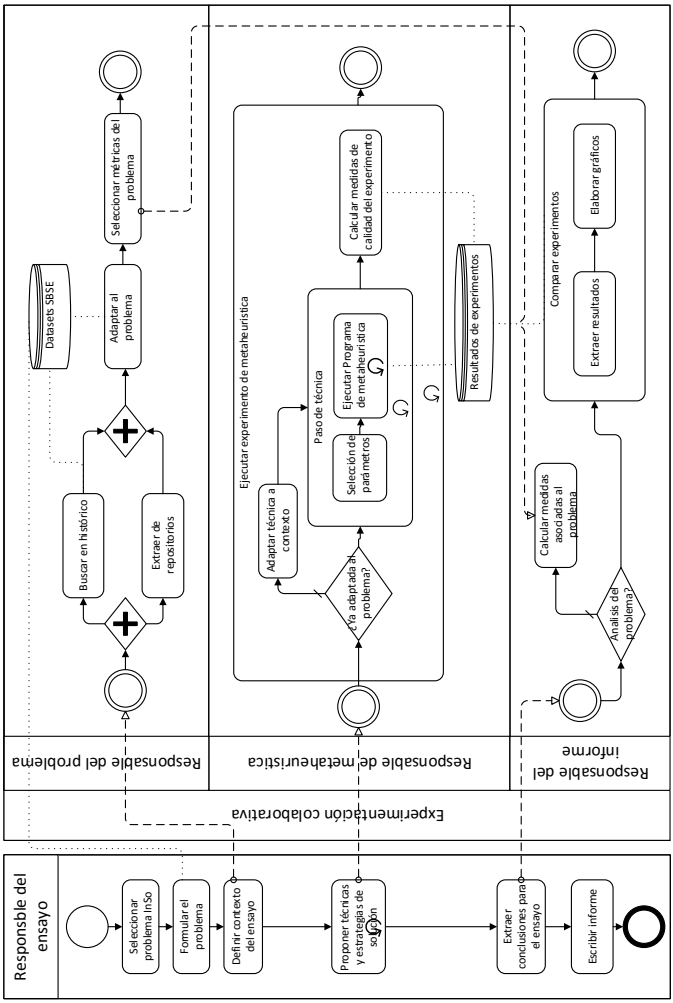


Figura 1: Flujo de trabajo para un ensayo en Ingeniería del software guiada por búsqueda

Cuando queremos aplicar una nueva metaheuristica y establecer su efectividad, necesitamos compararla bien contra un caso base, o bien contra los resultados obtenidos por otro conjunto de técnicas aplicadas en anteriores trabajos de investigación. Pero esta cuestión no está totalmente resuelta en la literatura. En primer lugar, algunas metaheurísticas tienen muchos parámetros que requieren un ajuste manual (ej. probabilidad de mutación o cruce, grado de voracidad), y en segundo lugar, porque son de naturaleza estocástica y necesitamos recurrir a profundos estudios estadísticos y de indicadores de calidad para poder hacer una comparación efectiva.
Si la metaheuristica no ha sido antes adaptada a ese problema, entonces habrá de serlo. Antes de poder compararla con los resultados de otras, se debe buscar cuál es la configuración que mejor resultado presenta. Esta selección de parámetros y la posterior comparativa se debe realizar en base a indicadores de calidad [35,21] y que son estrechamente dependientes de la estrategia de resolución (e.g. monobjetivo, multiobjetivo).

- *El nivel de generación de informes y extracción de conclusiones.* El análisis efectivo de los datos obtenidos durante el ensayo no es una tarea fácil. Debe abordarse, tanto desde el punto de vista de la metaheurística, como desde el punto de vista del problema y, en muchos casos, ampliar el análisis al contexto concreto del problema o conjunto de datos [3]. Además, en la mayoría de las ocasiones se deben abstraer los datos utilizando distintos tipos de gráficos y test estadísticos que permitan a los investigadores definir conclusiones que puedan compartirse en un formato unificado a toda la comunidad.

4. Caso del problema de la siguiente versión

En esta sección describimos como se instancia el conjunto de actividades genéricas mostradas en la figura 1 a un problema concreto de Ingeniería del Software (el problema de la siguiente versión) aplicando técnicas y algoritmos guiados por búsqueda.

Seleccionar el problema Nos centraremos en un problema ampliamente conocido en esta disciplina, el problema de la siguiente versión o NRP [6], para el que existen numerosos trabajos que ofrecen buenas soluciones utilizando distintas aproximaciones [23,1]. La limitación de recursos en los proyectos de desarrollo de software, obliga a dejar fuera de la siguiente versión de una aplicación ya construida algunas de las propuestas hechas por los clientes. Es decir, el equipo de desarrollo tiene que decidir cuál es el subconjunto de requisitos que tiene que contemplarse. Se busca combinar los métodos computacionales con la experiencia de los expertos humanos para obtener mejores conjuntos de requisitos de los que se producirían sólo mediante el juicio experto. Para ello este problema se formula como sigue.

Formular el problema Las selección de requisitos presenta variantes en su formulación, cada una destaca una parte del problema. Se definen requisitos y

clientes y se obliga a satisfacer por completo a los clientes seleccionando todas sus propuestas [6]; o se definen en la importancia de cada requisito individualmente [5], permitiendo la satisfacción parcial de los clientes. En cualquier caso, los requisitos presentan dependencias o interacciones entre ellos, imponiendo un orden de desarrollo determinado, pero la versión extensa que incorpora todos los tipos de dependencias entre requisitos no se formuló utilizando una representación en forma de grafo y matricial hasta el trabajo de del Sagrado et al. [25].

De esta manera para formular el problema, el responsable del ensayo podrá reutilizar una formulación ya propuesta o modificar o redefinir alguna de las existentes, generando un nuevo tipo de problema, por ejemplo, incluyendo el riesgo asociado a cada requisito como una dimensión más.

En nuestro caso, la formalización parte del conjunto de requsitos que aún no han sido desarrollados $\mathbf{R} = \{r_1, r_2, \ldots, r_n\}$ y que han sido propuestos por un conjunto de m clientes. Cada cliente i tiene una importancia para la empresa dentro del proyecto, $w_i \in \mathbb{R}$. Cada requisito $r_j \in \mathbf{R}$ tiene un coste c_j que recoge el esfuerzo de desarrollo. El valor del requisito r_j para el cliente i se representa con $v_{ij} \in \mathbb{R}$. La satisfacción, s_j, o valor añadido por la inclusión de r_j en la siguiente versión del software, se puede calcular como, $s_j = \sum_{i=1}^{m} w_i * v_{ij}$.

Si llamamos $X \subseteq R$ al conjunto de requisitos seleccionados, el coste y el valor de X vienen dados por las funciones:

$$esfuerzo(X) = \sum_{j,r_j \in X} c_j \quad \text{y} \quad satisfaccion(X) = \sum_{j,r_j \in X} s_j$$

respectivamente. En relación con la **estrategia**, consideraremos una versión multiobjetivo del problema que minimice el coste y maximice el valor del conjunto de requisitos seleccionados (i.e. buscaremos soluciones en un espacio de búsqueda de dos dimensiones).

Definir el contexto del ensayo Este proceso implica fijar el conjunto de datos sobre el que ejecutar las pruebas y, por lo tanto, entra dentro de las tareas del agente responsable del problema. En nuestra instancia se reutiliza el caso descrito por primera vez en [13] con el límite de esfuerzo fijado en 60. No obstante, puede que el conjunto de datos necesitase un pretratamiento, que ejecuta este agente, para trasformar el problema eliminando dependencias [25] o definir un nuevo caso extrayendo los datos de repositorios públicos de software para asignar errores como peticiones de nuevos requisitos [32].

Proponer técnicas y estrategias de solución Actualmente, se han propuesto numerosas técnica inteligentes para la solución del problema NRP con éxito. Bagnall et al. [6] mostró como aplicar ascenso de colinas, algoritmos voraces y recocido simulado teniendo en cuenta solo un tipo de dependencia. Otras soluciones alternativas se han basado fundamentalmente en algoritmos genéticos, [13] Del Sagrado et al. [24] adaptaron un sistema de colonia de hormigas para requisitos independientes y compararon los resultados con recocido simulado y un algoritmo genético. Estos algoritmos de colonia de hormigas también se han

utilizado contemplando sólo las dependencias de implicación [27]. Todas estas soluciones plantean una estrategia con un sólo objetivo, maximizar la satisfacción considerando el límite de esfuerzo como restricción. La consideración de todas las dependencias funcionales dentro del problema de búsqueda ha sido resuelta modelando las dependencias como un grafo dirigido [26,1].

Otros trabajos han aplicado una estrategia multiobjetivo con distintas técnicas metaheurísticas de naturaleza genética [34,9,8] y soluciones basadas en colonias de hormigas [25]. La visión multiobjetivo se ha usado para aplicar métodos exactos [30,14,3] o que reducen el espacio de búsqueda de las soluciones [32].

Para el ensayo que se está desarrollando en este caso vamos a estudiar la aplicabilidad de un algoritmo voraz [6] considerando todas las dependencias funcionales. Sus resultados se comparan con los obtenidos mediante una búsqueda aleatoria y con el frente de Pareto exacto, tras haber seleccionado un frente tipo de cada técnica aplicada.

Al tratarse de una nueva metaheurística, deberemos evaluar su rendimiento, en lo que se ha llamado *paso de técnica* en la figura 1. Aquí entraría en juego el ajuste de parámetros y la naturaleza estocástica de la técnica. Recurriremos al análisis de indicadores de calidad [35,21] de los resultados obtenidos en una serie de ejecuciones independientes de la técnica. Cada ejecución de la técnica nos proporcionará un frente de Pareto sobre el que calcularemos los indicadores de calidad: hipervolumen, extensión (*spread*), espaciado (*spacing*) y el número de soluciones. La distribución de los datos recopilados sobre estos indicadores servirá para darnos una idea del comportamiento individual de la técnica. Podemos ampliar el ámbito de comparación a otras técnicas comparando la distribución de indicadores calculados sobre el mismo número de ejecuciones independientes. Estos indicadores serán los que proporcionen el nivel de abstracción suficiente para permitir la comparabilidad de las técnicas.

En nuestro caso de estudio, vamos a comparar las soluciones obtenidas utilizando una técnica voraz, basada en la productividad $prod_j = s_j/c_j$ de cada requisito r_j, con las calculadas al realizar una búsqueda aleatoria. La tabla 1 recoge el resumen estadístico relativo a los indicadores de rendimiento utilizados para el caso de una técnica bi-objetivo, considerando 20 ejecuciones independientes.

Tabla 1: Indicadores para 20 ejecuciones (20 frentes)

	Algoritmo aleatorio				Algoritmo voraz			
	Hipervol	Spread	Spacing	Sols.	Hipervol	Spread	Spacing	Sols.
Min.	29956	0.60	0.25	27.00	30212	0.57	0.28	26.00
1st Qu.	30186	0.65	0.30	28.75	30529	0.61	0.34	28.00
Median	30292	0.68	0.31	30.50	30639	0.62	0.34	29.00
Mean	30274	0.69	0.31	30.65	30644	0.63	0.34	28.95
3rd Qu.	30333	0.72	0.33	32.00	30799	0.64	0.34	30.00
Max.	30541	0.83	0.36	35.00	31101	0.80	0.34	32.00

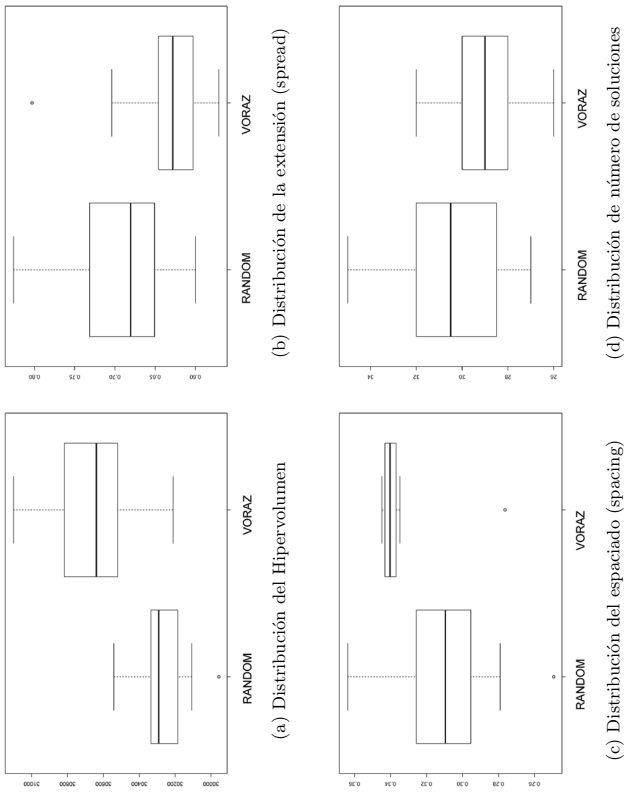


Figura 2: Comparativa gráfica de las distribuciones de los indicadores.

El comportamiento tipo de las técnicas se determina en base al valor de referencia que proporciona la mediana. Así, podemos guiarnos por este valor en los indicadores para seleccionar, de entre las ejecuciones realizadas, un frente de Pareto tipo para cada una de las técnicas aplicadas. La tabla 2 recoge los indicadores del frente de Pareto exacto y de los frentes tipo obtenidos tanto aleatoriamente, como aplicando la técnica voraz y cuya comparativa gráfica vemos en la figura 2.

Tabla 2: Indicadores asociados al frente exacto y a los frentes tipo

	Hipervol	Spread	Spacing	Sols.
Exacto	31165	0.587	0.341	31
Aleatorio	30339	0.692	0.315	31
Voraz	30637	0.630	0.345	29

La figura 3 muestra el frente exacto y los frentes tipo elegidos. Nos resta comprobar la similitud entre ellos. Esta tarea se aborda con la ayuda de test-no paramétricos sobre la distribución de la productividad de las soluciones en los frentes de Pareto. La hipótesis nula H_0 es que los datos de productividad de dos frentes de Pareto corresponden a poblaciones idénticas. Para comprobar la hipótesis aplicamos el test de Mann-Whitney-Wilcoxon para comparar las muestras independientes. En todos los casos, como el p-valor obtenido es mayor que el nivel de significancia de 0,05, aceptamos la hipótesis nula. Los datos de productividad asociados a los frentes de Pareto obtenidos con las distintas técnicas corresponden a poblaciones idénticas (ver tabla 3).

Tabla 3: Resultados del test del Wilcoxon

	Exacto-Aleatorio	Exacto-Voraz	Aleatorio-Voraz
p-value	0.2425	0.7899	0.1492

Después de obtener toda esta información sobre la técnica, si un investigador idease una nueva técnica, no sería necesario que volviese a realizar la implementación de las otras técnicas. En vez de eso, se centraría en ejecutar un número determinado de veces su técnica sobre el problema integrado en el flujo colaborativo, en obtener los indicadores y resultados, para, finalmente, compararlos con los disponibles. El flujo colaborativo agiliza la comparativa y descarga de trabajo al investigador.

Extracción de conclusiones y generación de informes Además de manejar y dar formato a los resultados generados directamente de las ejecuciones del software que implementa las técnicas, similares a las mostradas en la figuras 2 y 3 y las tablas, se deben acercar los resultados de los algoritmos de búsqueda al problema de Ingeniería del Software al que dan solución. Para ello, es necesario reinterpretar los resultados propuestos utilizando los términos y medias del dominio del problema, y así permitir que el ingeniero del software decida de entre las soluciones propuestas cuál es la más apropiada.

Esta trasferencia de los resultados a los proyectos software en desarrollo supone también la explotación de las técnicas de búsqueda en proyectos reales y se arbitra en base a indicadores de calidad dependientes de la formulación del problema. Para el caso que nos ocupa la generación de informes necesita cons-

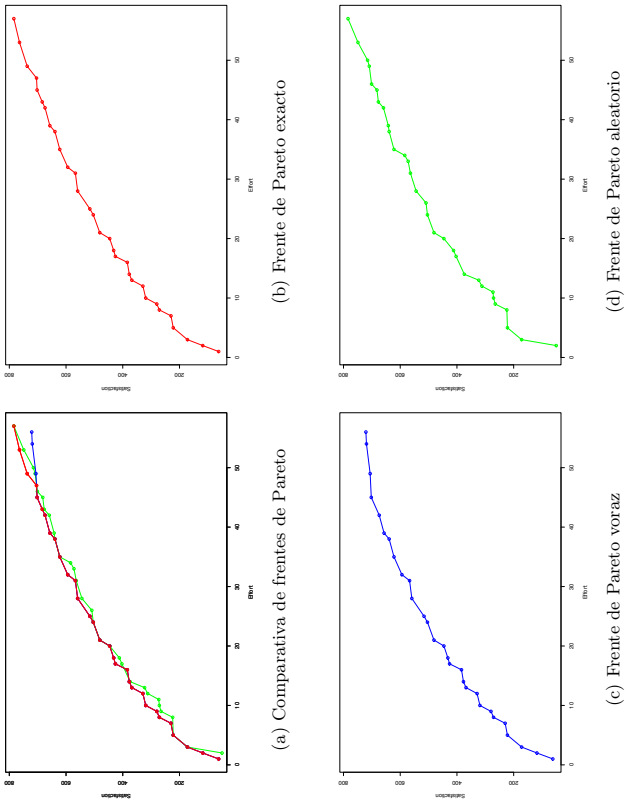


Figura 3: Frentes de Pareto.

truir indicadores de productividad, contribución y cobertura tanto de requisitos, como de clientes y soluciones [2].

Con todos estos datos disponibles el equipo de investigación responsable del ensayo está en disposición de generar un trabajo de carácter empírico para su difusión a la comunidad SBSE o a la comunidad científica en general.

5. Conclusiones

Hemos definido un flujo de trabajo unificado para la experimentación colaborativa basado en resultados e indicadores de calidad y rendimiento. Con ello proporcionamos un proceso común que ayuda a los investigadores a mantener un patrón estándar y que permite a los participantes de los ensayos de Ingeniería del Software guiada por búsqueda, colaborar en la comparación de los algoritmos y la reproducibilidad de los experimentos.

Para mostrar el uso del flujo de trabajo, se ha instanciado el conjunto de actividades genéricas al problema de la siguiente versión aplicando técnicas y algoritmos guiados por búsqueda simples.

Desde el punto de vista del equipo de investigación, una clara ventaja es la agilización de comparativas y la consiguiente descarga de trabajo. Así, un investigador que plantea una nueva técnica, no tiene que implementar otras técnicas como referencia y puede concentrarse en la configuración y ejecución de su propuesta sobre el problema generando solo indicadores y resultados que podrá comparar con el resto de resultados disponibles. No obstante, una necesidad inherente a este enfoque es la existencia de repositorios en los que los grupos de investigación compartan los resultados de experimentación, siguiendo los criterios comunes fijados en el flujo de trabajo unificado.

Como trabajo futuro planteamos la extensión el flujo de trabajo para cubrir también las tareas de explotación de los algoritmos de búsqueda en los proyectos software y la posiblidad de incorporarlas dentro de herramientas software que den soporte a la gestión del proyecto de desarrollo.

Agradecimientos. Este trabajo ha sido financiado parcialmente por la Universidad de Almería, la Red de Excelencia SEBASENet (TIN2015-71841-REDT) y por el Ministerio de Economía y Competitividad a través del proyecto TIN2013-46638-C3-1-P.

Referencias

1. del Águila, I., del Sagrado, Orellana, F.J.: Metaheurísticas como soporte a la selección de requisitos del software. In: Actas XVII Jornadas de Ingeniería del Software y Bases de Datos. SISTEDES, Almería, España (2012)
2. del Águila, I., del Sagrado, J., Bosch, A.: Análisis de las soluciones guiadas por búsqueda para el problema de selección de requisitos. In: XX Jornadas de Ingeniería del Software y Bases de Datos. SISTEDES (2015)
3. del Águila, I., del Sagrado, J., Chicano, F., Alba, E.: Resolviendo un problema multi-objetivo de selección de requisitos mediante resolutores del problema sat. In: XX Jornadas de Ingeniería del Software y Bases de Datos. SISTEDES (2015)
4. del Águila, I.M., del Sagrado, J.: Bayesian networks for enhancement of requirements engineering: a literature review. Requirements Engineering (may 2015)
5. van den Akker, J.M., Brinkkemper, S., Diepen, G., Versendaal, J.: Determination of the next release of a software product: an approach using integer linear programming. In: Proceeding of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ'05. pp. 247–262 (2005)

6. Bagnall, A.J., Rayward-Smith, V.J., Whittley, I.: The next release problem. Information & Software Technology 43(14), 883–890 (2001)
7. Boetticher, G.and Menzies, T., Ostrand, T.: Promise repository of empirical software engineering data. west virginia university, department of computer science. West Virginia University, Department of Computer Science (2007)
8. Durillo, J.J., Zhang, Y., Alba, E., Harman, M., Nebro, A.J.: A study of the bi-objective next release problem. Empirical Software Engineering 16(1), 29–60 (2011)
9. Finkelstein, A., Harman, M., Mansouri, S.A., Ren, J., Zhang, Y.: A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making. Requirement Engineering 14(4), 231–245 (2009)
10. Fomel, S.: Reproducible research as a community effort: Lessons from the Madagascar project. Computing in Science and Engineering 17(1), 20–26 (jan 2015)
11. Gómez, O.S., Juristo, N., Vegas, S.: Understanding replication of experiments in software engineering: A classification. Information and Software Technology 56(8), 1033–1048 (2014)
12. González-Barahona, J.M., Robles, G.: On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. Empirical Software Engineering 17(1-2), 75–89 (2012)
13. Greer, D., Ruhe, G.: Software release planning: an evolutionary and iterative approach. Information and Software Technology 46(4), 243–253 (Mar 2004)
14. Harman, M., Krinke, J., Medina-Bulo, I., Palomo-Lozano, F., Ren, J., Yoo, S.: Exact scalable sensitivity analysis for the next release problem. ACM Transactions on Software Engineering and Methodology (TOSEM) 23(2), 19 (2014)
15. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: Trends, techniques and applications. ACM Comput. Surv. 45(1), 11:1–11:61 (Dec 2012)
16. Juristo, N., Vegas, S.: The role of non-exact replications in software engineering experiments. Empirical Software Engineering 16(3), 295–324 (jun 2011)
17. Kastro, Y., Bener, A.B.: A defect prediction method for software versioning. Software Quality Journal 16(4), 543–562 (2008)
18. Menzies, T., Shepperd, M.: Special issue on repeatable results in software engineering prediction. Empirical Software Engineering 17(1-2), 1–17 (2012)
19. Misirli, A.T., Bener, A.B.: Bayesian networks for evidence-based decision-making in software engineering. IEEE Transactions on Software Engineering 40(6), 533–554 (2014)
20. Neumann, G., Swan, J., Harman, M., Clark, J.a.: The executable experimental template pattern for the systematic comparison of metaheuristics. Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion - GECCO Comp '14 pp. 1427–1430 (2014)
21. Okabe, T., Jin, Y., Sendhoff, B.: A critical survey of performance indices for multi-objective optimisation. In: Evolutionary Computation, 2003. CEC'03. The 2003 Congress on. vol. 2, pp. 878–885. IEEE (2003)
22. Parejo, J.A., Ruiz-Cortés, A., Lozano, S., Fernandez, P.: Metaheuristic optimization frameworks: A survey and benchmarking. Soft Computing 16(3), 527–561 (2012)
23. Pitangueira, A.M., Maciel, R.S.P., de Oliveira Barros, M.: Software requirements selection and prioritization using sbse approaches: A systematic review and mapping of the literature. Journal of Systems and Software 103, 267–280 (May 2015)
24. del Sagrado, J., del Águila, I., Orellana, F.: Ant colony optimization for the next release problem: A comparative study. In: Proceeding of Second International Sym-

posium on Search Based Software Engineering (SSBSE 2010), Benevento, Italy. pp. 67 –76 (sept 2010)
25. del Sagrado, J., del Águila, I.M., Orellana, F.J.: Multi-objective ant colony optimization for requirements selection. Empirical Software Engineering 20(3), 577–610 (2015)
26. del Sagrado, J., del Águila, I.M., Orellana, F.J.: Requirements interaction in the next release problem. In: Proceedings of 13th Annual Genetic and Evolutionary Computation Conference (GECCO 2011), Dublin, Ireland. pp. 241–242 (2011)
27. de Souza, J.T., Maia, C.L.B., do Nascimento Ferreira, T., do Carmo, R.A.F., Brasil, M.M.A.: An ant colony optimization approach to the software release planning with dependent requirements. In: Search Based Software Engineering, pp. 142–157. Springer (2011)
28. Swan, J., Adriaensen, S., Bishr, M., Burke, E.K., Clark, J.A., Causmaecker, P.D., Durillo, J., Hammond, K., Hart, E., Johnson, C.G., Kocsis, Z.A., Kovitz, B., Krawiec, K., Martin, S., Merelo, J.J., Minku, L.L., Ozcan, E., Pappa, G.L., Pesch, E., Garc, P., Schaerf, A., Sim, K., Smith, J.E., St, T., Voß, S., Wagner, S., Yao, X.: A Research Agenda for Metaheuristic Standardization. Mic pp. 1–3 (2015)
29. Tosun, A., Bener, A.B., Akbarinasaji, S.: A systematic literature review on the applications of Bayesian networks to predict software quality. Software Quality Journal pp. 1–33 (2015)
30. Veerapen, N., Ochoa, G., Harman, M., Burke, E.K.: An integer linear programming approach to the single and bi-objective next release problem. Information and Software Technology 65(0), 1 – 13 (2015)
31. Wen, J., Li, S., Lin, Z., Hu, Y., Huang, C.: Systematic literature review of machine learning based software development effort estimation models. Information and Software Technology 54(1), 41–59 (2012)
32. Xuan, J., Jiang, H., Ren, Z., Luo, Z.: Solving the large scale next release problem with a backbone-based multilevel algorithm. Software Engineering, IEEE Transactions on 38(5), 1195–1212 (2012)
33. Zhang, Y., M., H., Mansouri, A.: The sbse repository: A repository and analysis of authors and research articles on search based software engineering. *http* : *//crestweb.cs.ucl.ac.uk/resources/sbse*$_r$*epository/*. (2012)
34. Zhang, Y., Harman, M., Mansouri, S.A.: The multi-objective next release problem. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2007), London, England, UK. pp. 1129–1137 (2007)
35. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., Da Fonseca, V.G.: Performance assessment of multiobjective optimizers: an analysis and review. Evolutionary Computation, IEEE Transactions on 7(2), 117–132 (2003)

Un algoritmo híbrido para el problema NRP con interdependencias *

Francisco Palomo Lozano[1], Isabel M. del Águila[2] e Inmaculada Medina Bulo[1]

[1] Departamento de Ingeniería Informática, Universidad de Cádiz,
Avda. de la Universidad de Cádiz 10, 11519 Puerto Real, Spain
{francisco.palomo,inmaculada.medina}@uca.es
[2] Departamento de Informática, Universidad de Almería,
Ctra. de la Playa s/n, 04120 Almería, Spain
imaguila@ual.es

Resumen En este artículo presentamos un algoritmo híbrido para una variante del problema de la siguiente versión (NRP). En esta variante existe un conjunto de requisitos para los que se dispone de una estimación del esfuerzo necesario para su implementación y de la satisfacción percibida por los potenciales clientes con la inclusión de dichos requisitos. Entre estos requisitos existen relaciones de interdependencia, que establecen a ciertos requisitos como prerrequisitos de otros, o que obligan a implementar determinados requisitos simultáneamente en caso de incluirse alguno de ellos en la siguiente versión del producto a desarrollar. Dado un límite superior de esfuerzo prefijado, el objetivo es seleccionar un subconjunto de requisitos que cumpla todas las restricciones y maximice la satisfacción global de los clientes. La propuesta combina heurísticas con técnicas exactas. El rendimiento del algoritmo resultante en distintos escenarios realistas se compara con el de otras técnicas metaheurísticas previamente empleadas para el problema.

Keywords: Técnicas metaheurísticas, Ingeniería de requisitos, Problema de la siguiente versión, Interdependencias, NRP, SBSE.

1. Introducción

La selección de requisitos o características que debe poseer un producto entre las previamente definidas por los clientes es un proceso de gran importancia en proyectos de desarrollo de software que puede ser abordado con técnicas metaheurísticas y también con técnicas exactas [18]. En general, los requisitos presentan relaciones de interdependencia que recogen situaciones cotidianas en el desarrollo del software: requisitos que deben construirse en un orden concreto, que han de desarrollarse a la vez o que excluyen la posibilidad de desarrollar otros. El problema de la siguiente versión, o NRP, modela esta realidad.

* Parcialmente financiado por la red de excelencia TIN2015-71841-REDT (SEBASE-Net) y los proyectos TIN2014-60844-R (SAVANT) y TIN2015-65845-C3-3-R (DArDOS).

La principal dificultad estriba en que el problema computacional subyacente es complejo [16,15]. Sin embargo, dado su interés práctico para la industria del software, ha sido objeto de numerosos análisis desde este campo [14,3,11,2,4,20,21]. Un algoritmo híbrido ahorra tiempo de ejecución a costa de precisión, pero incluye una componente exacta que puede proporcionarle ventaja frente a otras técnicas aproximadas. Presentamos un algoritmo híbrido *determinista* para una variante del problema NRP basada en el modelo de van den Akker [2], en el que se considera un conjunto de requisitos para los que se dispone de una estimación del esfuerzo necesario para su implementación y de la satisfacción percibida por los potenciales clientes de incluirse dichos requisitos en la próxima versión del producto. Dicho modelo incluye también la posibilidad de que existan, como es común en la práctica, relaciones de interdependencia como las ya mencionadas.

El artículo se organiza como sigue. En la sección 2 se discuten los trabajos relacionados. A continuación, la sección 3 describe formalmente el problema y las distintas estrategias de resolución, con énfasis en las metaheurísticas. El algoritmo híbrido propuesto se discute en suficiente detalle en la sección 4. Una vez descrito, se evalúa su rendimiento en distintos escenarios y se comparan sus resultados con los obtenidos por las técnicas metaheurísticas de referencia en la sección 5. Para finalizar, en la sección 6 se resumen las conclusiones y se esbozan posibles líneas de trabajo futuro.

2. Trabajos relacionados

Es necesario aclarar que el problema NRP ha sido tratado profusamente en la literatura y que, no sólo la terminología, sino la propia definición del problema, pueden diferir de un autor a otro, por lo que existen numerosas variantes. Sin embargo, subyacen aspectos comunes en todas ellas y cabe decir que existen dos modelos principales, el de Bagnall y el de van den Akker, descritos a continuación, en los que se pueden incluir la mayoría.

Bagnall et al. [3] modeló el problema NRP como un problema de optimización. En su modelo se dispone de un conjunto de requisitos o mejoras que una compañía desea incluir en la siguiente versión de un producto de software, de los que se estima el coste de desarrollo, y de una serie de clientes que demandan la inclusión de ciertos requisitos en la siguiente versión del producto. Define variantes del problema, aunque nos referiremos a continuación a la más general.

Entre los requisitos pueden existir relaciones de dependencia; en particular, Bagnall considera que un requisito puede ser prerrequisito de otro, en cuyo caso la inclusión de este último exige la de aquel. En la estimación del coste de desarrollo de un requisito se presuponen satisfechos todos sus prerrequisitos, lo que permite estimar los costes por separado. En caso de que todos los requisitos exigidos por un cliente se incluyan, este aporta un beneficio a la compañía estimado según un peso o importancia que se asigna previamente al cliente. El objetivo es seleccionar un subconjunto de clientes a satisfacer, de forma que se maximice la suma de los pesos, o importancia total, y el coste total estimado de todos los requisitos a incluir en la siguiente versión no supere un presupuesto prefijado.

Van den Akker et al. [2] propuso un modelo alternativo en el que la importancia se asigna individualmente a cada requisito, con lo que se dispone de una forma de estimar su beneficio. El objetivo es seleccionar un subconjunto de requisitos a incluir en la siguiente versión que maximice el beneficio total y cuyo coste total estimado no exceda el presupuesto asignado. Al igual que en el modelo de Bagnall, en caso de existir interdependencias entre los requisitos, la solución propuesta debe respetarlas todas.

El modelo de van den Akker pone el foco en los requisitos, no en los clientes. No obstante, los clientes pueden incorporarse al modelo con la diferencia respecto al modelo de Bagnall de que, en este caso, se considera que pueden ser satisfechos parcialmente. En este sentido, cabe pensar que el modelo de van den Akker es más flexible. En presencia de clientes, puede obtenerse un modelo simple de estimación de beneficios para los requisitos si estos les asignan una importancia y se calcula para cada requisito su importancia ponderada con los pesos asignados a los clientes por la compañía.

Greer and Ruhe [11] trataron el problema de la planificación de versiones, importante en el contexto del desarrollo incremental de software. En este problema, se trata de distribuir los requisitos entre las distintas versiones planificadas teniendo en consideración las opiniones de los clientes, los recursos disponibles, las interdependencias y otros factores. El enfoque seguido consiste en resolver el problema a lo largo de una serie de iteraciones mediante un algoritmo genético.

Baker et al. [4] mostraron cómo los resultados que se obtienen mediante técnicas metaheurísticas relativos a la toma de decisiones en el contexto del problema NRP son superiores a los producidos por un experto.

Con algunas excepciones como [14,3,2,21], la mayor parte de la literatura se ha centrado en el desarrollo de técnicas metaheurísticas. En los trabajos de Sagrado et al. [20] y Pitangueira et al. [18] pueden encontrarse revisiones recientes de los trabajos relacionados que emplean dichas técnicas.

3. Formulación del problema y estrategias de resolución

Sea $R = \{r_1, \ldots, r_n\}$ un conjunto de requisitos propuestos por m clientes. Cada cliente i tiene distinta importancia para el proyecto, representada mediante un peso $w_i \in \mathbb{R}$. Todo $r_j \in R$ tiene asociado un esfuerzo de desarrollo e_j y un valor $v_{ij} \in \mathbb{R}$ para cada cliente i. La satisfacción, s_j, o valor añadido por la inclusión de r_j en la siguiente versión del software, se calcula como la suma ponderada $s_j = \sum_{i=1}^{m} w_i \cdot v_{ij}$. Se consideran una serie de interdependencias funcionales que pueden surgir entre los requisitos y que se definen como sigue:

1. *Implicación o precedencia* ($r_i \Rightarrow r_j$). El requisito r_j no puede ser incluido en el software sin incorporar r_i. Es decir, r_i es un prerrequisito de r_j.
2. *Combinación o acoplamiento* ($r_i \odot r_j$). Los requisitos r_i y r_j deben ser incluidos conjuntamente en el software.
3. *Exclusión* ($r_i \oplus r_j$). Los requisitos r_i y r_j no pueden ser incluidos conjuntamente en el software.

El objetivo del problema es encontrar un subconjunto de requisitos S que represente la mejor solución. Para ello es necesario construir S de forma que se maximice la satisfacción global de los clientes dentro de las limitaciones impuestas por los recursos asignados al proyecto y las interdependencias entre los requisitos. La satisfacción y el esfuerzo totales de S se definen como:

$$\text{sat}(S) = \sum_{r_j \in S} s_j \qquad \text{eff}(S) = \sum_{r_j \in S} e_j \tag{1}$$

y, formalmente, el problema consiste en:

$$\text{máx } \{\text{sat}(S) \mid \text{eff}(S) \leq B\} \tag{2}$$

siendo B el límite de esfuerzo.

3.1. Estrategias

A la hora de resolver un problema NRP que presenta interdependencias hay dos estrategias. La primera es tratarlas, bien dentro de los propios algoritmos, bien externamente. El tratamiento interno puede implicar, por ejemplo, adaptar la técnicas de búsqueda para que comprueben que no se producen conflictos [19]. El tratamiento externo puede realizarse, por ejemplo, traduciéndolas a lógica proposicional e incorporando resolutores SAT en el proceso [1]. La segunda consiste en transformar el problema original en otro que incluya únicamente requisitos independientes, tratando así de evitar las interdependencias. En el caso de las interdependencias de implicación, su eliminación fue tratada en el artículo original de Bagnall et al. [3], pero, en realidad, únicamente es efectiva para la versión más restringida del problema donde se debe satisfacer completamente al cliente.

En el trabajo de Sagrado et al. [20] se propone un método más general de eliminación de interdependencias para el problema NRP en el modelo de van der Akker. La idea es trasformar todas las interdependencias mediante la agrupación de requisitos en metarrequisitos, generando un nuevo escenario de metarrequisitos independientes. No obstante, existen ciertas limitaciones, pues en los grupos de requisitos que surgen al eliminar las interdependencias puede aparecer duplicados (un requisito puede formar parte de varios metarrequisitos).

Se describirán a continuación sucintamente las técnicas metaheurísticas que se compararán con el algoritmo híbrido propuesto en este trabajo. No consideraremos las dependencias de exclusión por las razones expuestas en [20], donde puede encontrarse una descripción más detallada de estas técnicas. Si bien cuando su número es pequeño puede resultar factible eliminarlas a cambio de resolver ejemplares extra, no se ha implementado aquí dicha posibilidad.

3.2. Metaheurísticas

A diferencia del algoritmo híbrido propuesto, las tres metaheurísticas que se presentan a continuación resuelven la versión biobjetivo del problema [20], pero en el contexto de este trabajo sus resultados se utilizan para maximizar un único objetivo, la satisfacción. Dado un conjunto de requisitos R, definimos:

1. La productividad:

$$p(R) = \sum_{r_i \in R} p(r_i) \quad \text{con} \quad p(r_i) = \frac{s_i}{e_i} \tag{3}$$

2. El conjunto de requisitos visibles, vis(R), que incluye todos los requisitos r_j que satisfacen las interdependencias y cumplen que $\text{eff}(R) + e_j \leq B$.

GRASP Greedy Randomized Adaptative Search Procedure (GRASP) [8] es una técnica iterativa que construye una solución voraz aleatoria y la mejora con búsqueda local. Se construye una lista de requisitos visibles ordenada por productividad. La búsqueda local es iterativa y trata de reemplazar la solución actual por una mejor de su vecindario. Es decir, se elimina de la solución el requisito con menor p y se selecciona otro de los visibles. Si la nueva solución mejora en términos de satisfacción de los clientes, sustituye a la anterior y se repite el proceso. GRASP termina cuando no hay una solución mejor en el vecindario.

Algoritmos genéticos Cada individuo se representa mediante un vector binario y define un conjunto de requisitos que satisface las restricciones del problema. Dado $[x_1, \ldots, x_n] \in \{0,1\}^n$, $x_i = 1$ si, y sólo si, el requisito $r_i \in S$. Cada generación representa la evolución de la población que, a través de las operaciones de cruce y mutación, puede producir nuevos individuos que poseen incluso mejores valores de satisfacción que los originales. Los operadores de cruce y mutación son operadores ciegos, en el sentido de que no garantizan ni que la solución satisfaga la restricción del límite de esfuerzo, ni las asociadas a las interdependencias. Por ello, se emplea un operador de reparación que elimina requisitos, en orden decreciente de productividad, hasta que se cumplen las restricciones [19].

Algoritmos de colonia de hormigas La optimización por colonia de hormigas emula el comportamiento de las hormigas cuando buscan el camino más corto desde su hormiguero hasta la comida [7]. Emplean una sustancia química, la feromona. Al elegir un camino, la hormiga adopta un comportamiento probabilístico. En cada encrucijada tiende a seguir el camino con más feromona, sustancia que segregan las hormigas conforme avanzan; si la cantidad de esta sustancia no es suficiente, elige al azar. La feromona se concentra en los caminos más transitados. El algoritmo utiliza un determinado número de hormigas, normalmente la mitad que requisitos. Cada una calcula una solución, comenzando en un lugar aleatorio y seleccionando los requisitos uno a uno. La hormiga localiza qué requisitos son visibles desde los que ya ha seleccionado y elige uno, combinando la información heurística (en este caso, la productividad) y la feromona. Finalmente, se actualizan los valores de la feromona de la mejor solución. Al construir la solución asociada a la hormiga k, a partir de un lugar i, se selecciona el próximo lugar, j, como sigue:

$$j = \begin{cases} \arg\ \text{máx}_{u \in N_i^k} [\tau_{ij}]^\alpha [\eta_{ij}]^\beta, & \text{si } q \leq q_0 \\ u, & \text{e.o.c.} \end{cases} \tag{4}$$

Así, si el último requisito seleccionado ha sido r_i, el siguiente será r_j. El valor q es un valor aleatorio entre $[0, 1]$, y $q_0 \in [0, 1]$ es un parámetro que controla el equilibrio entre intensificación y diversificación en el algoritmo, buscando la mejor cobertura del espacio de búsqueda. La probabilidad de que la hormiga k elija moverse de i a j es:

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{h \in N_i^k} [\tau_{ij}]^\alpha [\eta_{ij}]^\beta}, & \text{si } j \in N_i^k \\ 0, & \text{e.o.c.} \end{cases} \tag{5}$$

siendo la información heurística una media de productividad $\eta_{ij}^k = \mu \cdot p(r_j)$ con μ un valor de normalización y N_i^k el conjunto de requisitos visibles para la hormiga k. La mejor solución encontrada por las hormigas, S, actualiza el rastro de feromona τ_{ij} de acuerdo con:

$$(1 - \rho)\tau_{ij} + \rho \Delta\tau_{ij} \qquad \Delta\tau_{ij} = \frac{\text{sat}(S)}{\text{sat}(R)} \tag{6}$$

siendo $\rho \in [0, 1]$ la tasa de evaporación de la feromona.

4. Algoritmo híbrido para el problema NRP

El problema NRP es computacionalmente complejo y hereda su dificultad del problema de la mochila 0/1 (KP), que ha sido estudiado exhaustivamente por sus múltiples aplicaciones incluso desde antes de que existiera consciencia de este hecho [6]. La consecuencia principal es que, independientemente de la bondad de los algoritmos que se empleen, siempre exhibirán un bajo rendimiento para un número infinito de ejemplares del problema, en tanto en cuanto estos algoritmos sean exactos y $\mathcal{P} \neq \mathcal{NP}$ [10].

Por otro lado, en el caso particular de KP, existe un resultado negativo importante que nos impide encontrar algoritmos aproximados con rendimiento absoluto, esto es, que garanticen que el error absoluto en el que incurren esté acotado por una constante. Es decir, garantizar un error absoluto para KP es tan difícil computacionalmente como garantizar un resultado exacto.

Sin embargo, existen algoritmos eficientes para KP que garantizan que el error relativo que cometen está acotado por una constante. Así, existen algoritmos $(1-\epsilon)$-aproximados que producen soluciones cuyo valor es, al menos, $1 - \epsilon$ veces el del máximo alcanzable. Esto es, el error relativo que poseen las soluciones aproximadas que producen respecto a las óptimas es, a lo sumo, ϵ. Lógicamente, su tiempo de ejecución depende de ϵ: cuanto menor sea ϵ, mayor será el tiempo de ejecución. Aunque para KP es posible disminuir ϵ aumentando el tiempo de ejecución, en el peor caso, en un polinomio en el tamaño de la entrada y ϵ^{-1} [13], estas garantías desaparecen cuando se introducen dependencias en el problema, como en el caso del problema NRP que nos ocupa. Considérese al respecto la noción de $\mathcal{NP}$-completitud en sentido estricto [9].

En tal caso, conforme se reduce el error relativo que se desea garantizar, la eficiencia de los algoritmos disponibles se degrada rápidamente, incluso aunque se mantenga el tamaño de los ejemplares. Existe pues, en general, un compromiso entre tiempo y precisión. Es posible sacrificar precisión, renunciando a obtener siempre una solución óptima, a cambio de disminuir el tiempo de ejecución, o invertir tiempo de ejecución en obtener mejores soluciones.

Un algoritmo híbrido, que combine un algoritmo exacto con heurísticas, representa un compromiso entre ambas opciones, pero posee el potencial de producir mejores soluciones que otros algoritmos puramente heurísticos o metaheurísticos a un coste razonable. Para que este enfoque sea factible, la parte exacta del algoritmo debe emplearse para resolver solo una parte concreta del problema o una versión simplificada del mismo, de forma que a partir del resultado se pueda recuperar una solución aproximada. El algoritmo híbrido propuesto en el presente trabajo consta de las siguientes etapas:

1. Transformación de los requisitos en metarrequisitos.
2. Resolución del problema NRP sobre los metarrequisitos.
3. Reparación de la solución para eliminar requisitos duplicados.
4. Resolución recursiva sobre los requisitos no incluidos en la solución.

El algoritmo ha sido cuidadosamente implementado en C++ estándar. A continuación se discute cada etapa y se presentan fragmentos de pseudocódigo, aunque un buen número de detalles y mejoras obvias se han omitido por claridad expositiva.

4.1. Transformación en metarrequisitos

La transformación de los requisitos en metarrequisitos se basa en [20] y tiene en consideración las interdependencias de precedencia y de combinación.

En primer lugar, el algoritmo 1 inicializa una matriz de adyacencia de $n \times n$ con la información proporcionada por las parejas de requisitos que aparecen en cada relación de interdependencia. Todas las entradas de la matriz booleana a reciben primero el valor falso ($\bot$) y cambiarán a verdadero ($\top$) según sean o no afectadas por alguna interdependencia. En el caso de las interdependencias de combinación, la relación es simétrica, lo que debe tenerse en cuenta al rellenar las entradas correspondientes.[3] En el caso de la relación de precedencia, se invierten las parejas. El resultado es la matriz de la relación $\odot \cup \Rightarrow^{-1} = \odot \cup \Leftarrow$, donde $\Leftarrow$ representa la inversa de la relación de precedencia, es decir, $r_i \Leftarrow r_j$ si r_j es prerrequisito de r_i. En lugar de trabajar directamente con los requisitos, el algoritmo emplea sus índices.

[3] En los conjuntos de datos disponibles para experimentación solo se especifica uno de los dos pares ordenados correspondientes a cada par de requisitos relacionados, lo que resulta conveniente por brevedad.

```
1  for i ← 1 to n do
2      for j ← 1 to n do
3          a[i, j] ← ⊥
4  ▷ Relación de combinación.
5  for all (i, j) ∈ ⊙ do
6      ▷ Simétrica.
7      a[i, j] ← ⊤
8      a[j, i] ← ⊤
9  ▷ Relación de precedencia.
10 for all (i, j) ∈ ⇒ do
11     ▷ Inversa.
12     a[j, i] ← ⊤
13 return a
```

Algoritmo 1: Inicialización.

A continuación, el algoritmo 2 proporciona el cierre reflexivo y transitivo de la relación resultante mediante una sencilla modificación del algoritmo de Warshall. Por consiguiente, el resultado es la matriz de la relación $(\odot \cup \Leftarrow)^*$. A partir de esta matriz puede determinarse en tiempo constante si un requisito depende de otro, y extraerse fácilmente todos los requisitos de los que depende uno dado.

```
1  ▷ Cierre transitivo.
2  for k ← 1 to n do
3      ▷ Cierre reflexivo.
4      a[k, k] ← ⊤
5      for i ← 1 to n do
6          ▷ Mejora.
7          if a[i, k] then
8              for j ← 1 to n do
9                  a[i, j] ← a[i, j] ∨ a[k, j]
10 return a
```

Algoritmo 2: Cierre reflexivo y transitivo.

Esta modificación incorpora una ligera, pero efectiva, mejora. La matriz a se representa internamente de forma muy compacta (1 bit por entrada), lo que permitiría realizar otras mejoras, como procesar cada fila mediante operaciones de bits muy eficientes.

Finalmente, el proceso de obtención del conjunto de metarrequisitos se ilustra en el algoritmo 3. Internamente, cada metarrequisito contiene el índice del requisito a partir del que se genera, así como el esfuerzo y satisfacción totales de todos los requisitos que lo integran. Cada requisito produce un metarrequisito. Un requisito sin dependencias da lugar a un metarrequisito unitario, que solo lo contiene a él mismo. Si un metarrequisito posee idéntica composición que otro previamente generado, se elimina, aunque para facilitar la comprensión esto no se ha reflejado en el algoritmo.

```
1  M
2  for i    1 to n do
3      ▷ Requisitos de los que depende r_i.
4      T    ∅
5      for k ← 1 to n do
6          if a[i, k] then
7              T    T ∪ {r[k]}
8      ▷ Incluye el nuevo metarrequisito.
9      M    M ∪ {⟨i, eff(T), sat(T)⟩}
10 return M
```

Algoritmo 3: Metarrequisitos.

4.2. Resolución sobre los metarrequisitos

La técnica de programación dinámica permite obtener una familia importante de *algoritmos pseudopolinómicos* para KP. Por ejemplo, sea S un conjunto de objetos, $c(x)$ y $b(x)$ el coste y beneficio de un objeto $x \in S$, y B el presupuesto disponible. La siguiente ecuación de Bellman puede resolverse mediante programación dinámica por costes, proporcionando una solución óptima para KP a un coste razonable cuando el número de objetos y el presupuesto son moderados:

$$\begin{aligned} &z(\varnothing, B) = 0 \\ &z(S, B) = z(S \setminus \{x\}, B) && \text{si } c(x) > B \\ &z(S, B) = \text{máx}\{z(S \setminus \{x\}, B), z(S \setminus \{x\}, B - c(x)) + b(x)\} && \text{si } c(x) \leq B \end{aligned} \quad (7)$$

Esto puede traducirse fácilmente al dominio del problema NRP (si se eliminan las interdependencias), cambiando objetos por requisitos, coste por esfuerzo, beneficio por satisfacción y presupuesto por límite de esfuerzo. Así, en la ecuación 7, el conjunto S representaría una colección finita de requisitos, x un requisito arbitrario de S con esfuerzo $c(x)$ y satisfacción $b(x)$, B el límite de esfuerzo y z la máxima satisfacción que se puede alcanzar con cualquier subconjunto de S sin exceder el límite de esfuerzo.

Para la resolución del problema NRP sin interdependencias se ha desarrollado una versión muy eficiente del algoritmo de Nemhauser-Ullmann, que emplea programación dinámica por costes [17]. Este algoritmo es exacto, por lo que siempre produce soluciones óptimas y, a diferencia de otros, no impone restricciones adicionales sobre su entrada. Esta versión se presenta en el algoritmo 4. Su buen rendimiento radica, en parte, en el empleo de *funciones escalera*. Una función escalera no es más que una función creciente con un número finito de peldaños o rellanos. Estas funciones se pueden representar eficientemente como listas de pares, que contienen las coordenadas de cada peldaño, con cuatro operaciones: *cero*, *desplazamiento*, *máximo* y *aplica*. Su funcionamiento se explica en más detalle en [12].

```
1   ▷ Calcula las satisfacciones máximas.
2   m[0] ← cero()
3   for k ← 1 to n do
4       f     desplazamiento(m[k − 1], eff(r[k]), sat(r[k]))
5       m[k] ← máximo(m[k − 1], f)
6   ▷ Recupera la solución óptima calculada.
7   S     ∅
8   for k ← n to 1 do
9       if aplica(m[k], B) ≠ aplica(m[k − 1], B) then
10          S     S ∪ {k}
11          B     B − eff(r[k])
12  return S
```

Algoritmo 4: Nemhauser-Ullmann para NRP sin interdependencias.

Desde la publicación de este algoritmo en 1969, varios autores habían constatado que parecía comportarse bien en la práctica cuando se resolvían ejemplares aleatorios del problema KP, a pesar de tratarse de un problema $\mathcal{NP}$-difícil, sin existir una justificación teórica que soportara estas observaciones. Sin embargo, en 2004, Beier and Vöcking proporcionaron una explicación del comportamiento del algoritmo, bajo hipótesis muy generales, mediante técnicas de *análisis suave*. Queda fuera del ámbito del presente trabajo discutir los aspectos, muy técnicos, de estos resultados [5], pero pueden resumirse diciendo que bajo suposiciones razonables sobre las distribuciones de coste y valor de los objetos implicados, el tiempo de ejecución y el espacio consumido esperados son polinómicos en el número de objetos.[4]

4.3. Reparación de la solución

Debido al proceso de transformación, un requisito puede formar parte de varios metarrequisitos. Por tanto, la solución S puede estar formada por metarrequisitos en los que puede aparecer varias veces un mismo requisito. La presencia de repeticiones indica que la solución obtenida en la etapa anterior puede ser subóptima. De ahí que aunque el algoritmo empleado sea exacto para NRP sin interdependencias, su combinación con la etapa previa de transformación que permite resolver el problema NRP original con interdependencias sea heurística.

Es necesario sustituir los metarrequisitos por sus requisitos para obtener una solución al problema original, pero sin que estos aparezcan repetidos. Para ello se construye un vector f con la frecuencia de aparición de cada requisito en los metarrequisitos. Este vector permite expandir eficientemente los metarrequisitos contenidos en S evitando las repeticiones. Al final de la etapa de reparación, la solución S contendrá la unión de todos los requisitos previamente contenidos en sus metarrequisitos.

[4] El algoritmo es pseudopolinómico en el peor caso, como se desprende de un análisis ordinario, y, bajo las hipótesis de suavidad correspondientes, polinómico.

4.4. Resolución recursiva

Una vez que disponemos del conjunto de requisitos solución S, el conjunto $R' = R \setminus S$ contiene aquellos requisitos que no se han incluido en ella. Debido a la necesidad de reparar la solución en la etapa anterior, es posible que el límite de esfuerzo sobrante $B' = B - \text{eff}(S)$ pueda emplearse para incluir requisitos de R' en la solución. No obstante, es necesario asegurarse de que estos requisitos siguen satisfaciendo las restricciones de interdependencia.

El algoritmo 5 permite reducir las interdependencias obviando aquellos pares que los requisitos de S satisfacen. Para ello basta emplear el vector de frecuencias generado en la etapa anterior. Únicamente pervivirán los pares en los que ninguna componente corresponda a un requisito de S, es decir, aquellos en los que la frecuencia sea 0 en ambos extremos.

```
1  ▷ Nueva relación de combinación.
2  C ← ∅
3  for all (i, j) ∈ ⊙ do
4      if f[i] = 0 ∧ f[j] = 0 then
5          C ← C ∪ {(i, j)}
6  ▷ Nueva relación de precedencia.
7  P ← ∅
8  for all (i, j) ∈ ⇒ do
9      if f[i] = 0 ∧ f[j] = 0 then
10         P ← P ∪ {(i, j)}
11 return ⟨C, P⟩
```

Algoritmo 5: Reducción.

Finalmente, únicamente queda resolver recursivamente el problema para R', con límite de esfuerzo B' e interdependencias reducidas C y P. La solución al problema reducido se une a S, salvo que dicha solución sea $\varnothing$, en cuyo caso el proceso recursivo termina y S es la solución final.

5. Resultados experimentales

Los conjuntos de datos empleados siguen el modelo de van den Akker [2] para el problema NRP con interdependencias. El primero [11] consta de 20 requisitos propuestos por 5 clientes. El segundo [19] ha sido generado aleatoriamente y cuenta con 100 requisitos propuestos por 5 clientes. Los valores de esfuerzo se fijan entre 1 y 20 días de trabajo y los pesos de clientes y valores asignados por estos a los requisitos pueden entenderse como los siguientes niveles linguísticos: sin importancia (1), poca importancia (2), importante (3), muy importante (4) y extremadamente importante (5).

Los escenarios de experimentación para cada conjunto de datos se han seleccionado estableciendo unos límites de esfuerzo relativos al esfuerzo total necesario para implementar todos los requisitos. Para cada conjunto de datos, los valores

limites son el 30 %, el 50 % y el 70 % del esfuerzo total y se recogen, junto al número de requisitos y de interdependencias de cada tipo, en la tabla 1.

Tabla 1. Características de los conjuntos de datos y escenarios.

	1.er conjunto			2.º conjunto		
B	30 %	50 %	70 %	30 %	50 %	70 %
	25	43	60	312	519	726
$\|R\|$	20			100		
$\|\Rightarrow\|$	8			38		
$\|\odot\|$	2			4		

Para cada escenario, las tablas 2 y 3 contienen las satisfacciones máximas, los esfuerzos con los que se alcanzan y los tiempos de ejecución de las correspondientes técnicas. Respecto a las metaheurísticas, para cada escenario se han realizado 100 ejecuciones consecutivas con distintas configuraciones de los parámetros propios de cada una, como pueden ser las probabilidades de cruce o mutación en NSGA II o el grado de voracidad en GRASP. Los datos mostrados en las tablas son los valores de la media de los experimentos con las mejores configuraciones establecidas en trabajos previos [20]. Los parámetros utilizados en los algoritmos se han seleccionado para conseguir en torno a 10 000 ejecuciones de las funciones de evaluación. En cambio, el algoritmo híbrido es determinista y, aunque solo es necesaria una ejecución por cada escenario, se han realizado varias al efecto de estimar mejor su tiempo de ejecución.

Tabla 2. Satisfacción máxima alcanzada y esfuerzo empleado para ello.

	1.er conjunto						2.º conjunto					
	30 %		50 %		70 %		30 %		50 %		70 %	
	sat	eff	sat	eff	sat	eff	sat	eff	sat	eff	sat	eff
ACS	516	25	626	40	689	60	1239	307	1635	513	2034	726
NSGA II	461	21	567	39	607	48	1139	295	1666	509	2100	718
GRASP	508	25	602	43	689	60	1157	311	1524	519	1984	720
Híbrido	504	24	684	43	783	57	1275	311	1805	519	2247	725

Tabla 3. Tiempo (ms) medio empleado para alcanzar la solución.

	1.er conjunto			2.º conjunto		
	30 %	50 %	70 %	30 %	50 %	70 %
ACS	639,10	787,62	836,76	616873,55	770221,22	881950,91
NSGA II	1891,55	1980,93	2034,25	28127,77	35045,76	38295,95
GRASP	362,80	1208,25	839,84	28915,30	118336,00	324604,00
Híbrido	4,00	6,00	13,00	37,00	40,00	28,00

El algoritmo híbrido es superior en la práctica totalidad de los escenarios y es capaz de encontrar soluciones más satisfactorias, invirtiendo el esfuerzo sobrante en ello. Las técnicas metaheurísticas, en cambio, a menudo no son capaces de encontrar estas soluciones en los extremos del frente de Pareto óptimo. GRASP obtiene sus mejores resultados con el mayor grado de aleatoriedad, 0,9 en el valor de parámetro que controla la aleatoriedad frente a la voracidad, favoreciendo la exploración del espacio de búsqueda. NSGA II utiliza una probabilidad de mutación $1/n$ para n requisitos. Sus mejores resultados los obtuvo con los mayores tamaños de población, siempre manteniendo en 10 000 el número de ejecuciones de la función de evaluación. ACS utiliza $n/2$ hormigas, que en el inicio se colocan aleatoriamente en requisitos visibles distintos. Sus mejores resultados se obtuvieron teniendo en cuenta tanto la información heurística como la feromona, pero dando mayor importancia a la heurística.

Respecto a los tiempos consignados en la tabla 3 conviene advertir que los de las metaheurísticas se calcularon con un Intel Pentium 4 a 3,20 GHz y los del algoritmo híbrido con un Intel Core i5 5200U a 2,20 GHz. En ambos casos se limitó la memoria a 1 GiB y la ejecución a un solo núcleo. No obstante, aunque las máquinas son diferentes, es evidente que los tiempos normalizados son muy inferiores en todos los escenarios.

6. Conclusiones y trabajo futuro

Se ha presentado un nuevo algoritmo para la resolución del problema NRP con interdependencias funcionales. Este algoritmo es híbrido, pues incorpora un algoritmo exacto para resolver un problema transformado sin interdependencias y una heurística que permite reducir el problema original al transformado. Posteriormente a la resolución del problema transformado, el algoritmo produce una solución al problema original, quizás tras resolver una serie de problemas subsidiarios más sencillos.

Se ha evaluado el algoritmo propuesto con conjuntos de datos empleados previamente en la literatura de referencia y comparado con tres técnicas heurísticas. Los resultados preliminares son favorables en la práctica totalidad de los escenarios, exhibiendo no únicamente mejoras en las soluciones obtenidas mediante las otras técnicas, sino además una importante reducción del tiempo de ejecución necesario. El algoritmo híbrido es, en esencia, determinista, por lo que no requiere ajuste de parámetros ni ejecuciones adicionales.

Entre las posibles líneas de trabajo futuro cabe destacar la realización de un estudio estadístico detallado, la ampliación del número de experimentos, la consideración de otras interdependencias aparte de las de precedencia y combinación, y la comparación del algoritmo híbrido con otras técnicas, tanto heurísticas como exactas.

Referencias

1. del Águila, I.M., del Sagrado, J., Chicano, F., Alba, E.: Resolviendo un problema multi-objetivo de selección de requisitos mediante resolutores del problema sat. In:

XX Jornadas de Ingeniería del Software y Bases de Datos. SISTEDES (2015)
2. van den Akker, J.M., Brinkkemper, S., Diepen, G., Versendaal, J.: Determination of the next release of a software product: an approach using integer linear programming. In: Proceeding of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ'05. pp. 247–262 (2005)
3. Bagnall, A.J., Rayward-Smith, V.J., Whittley, I.: The next release problem. Information & Software Technology 43(14), 883–890 (2001)
4. Baker, P., Harman, M., Steinhöfel, K., Skaliotis, A.: Search based approaches to component selection and prioritization for the next release problem. In: 22nd IEEE International Conference on Software Maintenance (ICSM 2006), 24-27 September 2006, Philadelphia, Pennsylvania, USA. pp. 176–185 (2006)
5. Beier, R., Vöcking, B.: Random knapsack in expected polynomial time. Journal of Computer and System Sciences 69(3), 306–329 (2004)
6. Dantzig, G.B.: Discrete variable extremum problems. Operations Research 5, 266–277 (1957)
7. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 26(1), 29–41 (1996)
8. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. Journal of global optimization 6(2), 109–133 (1995)
9. Garey, M.R., Johnson, D.S.: Strong NP-completeness results: Motivation, examples, and implications. Journal of ACM 25, 499–508 (1978)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. W. H. Freeman (1979)
11. Greer, D., Ruhe, G.: Software release planning: an evolutionary and iterative approach. Information and Software Technology 46, 243–253 (2004)
12. Harman, M., Krinke, J., Medina-Bulo, I., Palomo-Lozano, F., Ren, J., Yoo, S.: Exact scalable sensitivity analysis for the next release problem. ACM Transaction on Software Engineering Methodology 23(2), 19:1–19:31 (2014)
13. Hochbaum, D.S. (ed.): Approximation Algorithms for NP-hard Problems, chap. 9, pp. 346–398. PWS Publishing Company (1997)
14. Jung, H.W.: Optimizing value and cost in requirements analysis. IEEE Software 15(4), 74–78 (1998)
15. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer (2004)
16. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Willey & Sons (1990)
17. Nemhauser, G.L., Ullmann, Z.: Discrete dynamic programming and capital allocation. Management Science 15, 494–505 (1969)
18. Pitangueira, A.M., Maciel, R.S.P., de Oliveira Barros, M.: Software requirements selection and prioritization using SBSE approaches: A systematic review and mapping of the literature. Journal of Systems and Software 103, 267–280 (May 2015)
19. del Sagrado, J., del Águila, I.M., Orellana, F.J.: Requirements interaction in the next release problem. In: Proceedings of the 13th annual conference companion on genetic and evolutionary computation. pp. 241–242. ACM (2011)
20. del Sagrado, J., del Águila, I.M., Orellana, F.J.: Multi-objective ant colony optimization for requirements selection. Empirical Software Engineering 20(3), 577–610 (2015)
21. Veerapen, N., Ochoa, G., Harman, M., Burke, E.K.: An integer linear programming approach to the single and bi-objective next release problem. Information and Software Technology 65, 1–13 (2015)

Dos estrategias de búsqueda *anytime* basadas en programación lineal entera para resolver el problema de selección de requisitos

Francisco Chicano[1], Miguel Ángel Domínguez[1], Isabel del Águila[2], José del Sagrado[2] y Enrique Alba[1]

[1] Universidad de Málaga, Andalucía Tech, España
chicano@lcc.uma.es, miguel.angel.dominguez.rios@uma.es, eat@lcc.uma.es
[2] Universidad de Almería, España
{imaguila,jsagrado}@ual.es

Resumen El problema de selección de requisitos (o *Next Release Problem*, NRP) consiste en seleccionar el subconjunto de requisitos que se va a desarrollar en la siguiente versión de una aplicación software. Esta selección se debe hacer de tal forma que maximice la satisfacción de las partes interesadas a la vez que se minimiza el esfuerzo empleado en el desarrollo y se cumple un conjunto de restricciones. Trabajos recientes han abordado la formulación bi-objetivo de este problema usando técnicas exactas basadas en resolutores SAT y resolutores de programación lineal entera. Ambos se enfrentan a dificultades cuando las instancias tienen un gran tamaño. En la práctica, no es necesario calcular todas las soluciones del frente de Pareto (que pueden llegar a ser muchas) y basta con obtener un buen número de soluciones no dominadas bien distribuidas en el espacio objetivo. Las estrategias de búsqueda basadas en ILP que se han utilizado en el pasado para encontrar un frente bien distribuido en cualquier instante de tiempo solo buscan soluciones que pueden obtenerse minimizando una suma ponderada de los objetivos (*soluciones soportadas*). En este trabajo proponemos dos estrategias basadas en ILP que son capaces de encontrar el frente completo con suficiente tiempo y que, además, tienen la propiedad de aportar un conjunto de soluciones bien distribuido en el frente objetivo en cualquier momento de la búsqueda.

1. Introducción

La Ingeniería de Requisitos define el proceso, o conjunto de tareas, para descubrir el propósito de cualquier sistema, identificando las personas involucradas y sus necesidades [10]. Este proceso es esencial en el desarrollo de sistemas software, ya que debido a la naturaleza lógica del software, la principal medida de éxito del sistema desarrollado vendrá dada por el grado de consecución o cumplimiento de los requisitos. Los procesos relacionados con los requisitos no son fáciles de llevar a cabo porque residen en el espacio del problema y no en el de la solución, además son procesos difusos en el ciclo de vida de desarrollo

de software, en el que ideas informales deben ser traducidas a ideas formales; clientes y organizaciones deben colaborar para alcanzar un acuerdo preciso y sin ambigüedades de lo que debe ser desarrollado.

La planificación de versiones, es decir, la definición de cómo va a ir evolucionando un producto software a lo largo de su vida útil, es una tarea fundamental ligada al campo de la Ingeniería de Requisitos. En general, los clientes proponen numerosas nuevas características o requisitos que en la mayoría de los casos no todas pueden completarse dentro de las limitaciones impuestas por el tiempo y los recursos y que, por tanto, deben acotarse de alguna manera [2]. Estas decisiones suelen tener que considerar varios objetivos diferentes e incluso conflictivos, como las interacciones o dependencias entre las características candidatas, las preferencias de los clientes o las limitaciones en los recursos. En otras palabras, la planificación de versiones, en general, implica la optimización basada en múltiples criterios [7]. Los clientes, que buscan su propio interés, demandan las mejoras que consideran importantes, pero no todas ellas pueden ser satisfechas. Por un lado, cada requisito significa un coste en términos de esfuerzo que la empresa tiene que asumir, y por otro lado, ni todos los clientes son igualmente importantes para la empresa, ni todas las características son igualmente importantes para los clientes. Los factores de mercado también pueden influir en este proceso de selección: la empresa puede estar interesada en satisfacer las necesidades de los clientes más nuevos o en garantizar que cada cliente ve cumplido al menos uno de sus requisitos propuestos.

Este problema de planificación multi-objetivo se ha resuelto en el pasado usando tanto metaheurísticas [11], que no aseguran la calidad de las soluciones obtenidas pero son capaces de obtener soluciones de calidad aceptable en tiempos cortos, como técnicas exactas que calculan el frente óptimo de Pareto [1,13]. En este trabajo nos acercamos a este segundo enfoque. En particular, hemos observado que en el trabajo de Veerapen et al. [13] el algoritmo empleado para encontrar el frente completo requiere mucho tiempo para las instancias grandes, mientras que el algoritmo que utilizan para encontrar soluciones no dominadas bien distribuidas en el espacio objetivo sólo es capaz de encontrar soluciones soportadas[3]. En este trabajo proponemos dos estrategias (descritas en las secciones 4.4 y 4.5) que tratan de encontrar soluciones del frente que estén bien distribuidas, pero sin renunciar a la completitud, es decir, con suficiente tiempo, los algoritmos pueden calcular el frente de Pareto completo. Además, comparamos estas estrategias con otras tres más basadas en programación lineal entera y con técnicas metaheurísticas. Además de la propuesta de las dos estrategias de búsqueda, respondemos las siguientes preguntas de investigación:

- **RQ1**: ¿Cuál es la calidad de la parte del frente de Pareto encontrada por los distintos algoritmos comparados cuando limitamos el tiempo de ejecución de los mismos?
- **RQ2**: ¿Cuándo conviene utilizar metaheurísticas para resolver el problema y cuándo es mejor usar algoritmos basados en programación lineal entera?

[3] Se denominan *soluciones soportadas* a aquellas que pueden obtenerse minimizando una suma ponderada de los objetivos.

Para responder a ellas hemos realizado un estudio experimental con 19 instancias del problema y un total de ocho algoritmos diferentes.

El resto del artículo está organizado como sigue. En la sección 2 formalizamos el problema de la siguiente versión y la sección 3 presenta los programas lineales enteros usados para resolverlo. En la sección 4 presentamos los distintos algoritmos basados en ILP que utilizamos. La sección 5 presenta los resultados de un estudio experimental realizado para comparar los distintos algoritmos. Por último, la sección 6 presenta las conclusiones y el trabajo futuro.

2. El problema de la siguiente versión

Partimos de un conjunto de requisitos con dependencias $R = \{r_1, r_2, \ldots, r_n\}$ que aún no han sido desarrollados y que han sido propuestos por un conjunto de m clientes. Cada cliente i tiene un peso asociado $w_i \in \mathbb{R}$ que mide su importancia dentro del proyecto. Cada requisito $r_j \in R$ tiene un coste c_j para la empresa, es decir cada r_j consumirá c_j recursos si se desarrolla. El mismo requisito puede ser sugerido por varios clientes y su *valor* puede ser diferente para cada uno de ellos. El valor del requisito r_j para el cliente i se representa con $v_{ij} \in \mathbb{R}$.

En este punto cabe mencionar dos formas de valorar un conjunto de requisitos seleccionado. Por un lado, Xuan et al. [14] consideran que un cliente *está satisfecho* sólo cuando todos sus requisitos se implementan y, en este caso, sumamos su peso w_i al *grado de satisfacción* asociado al conjunto de requisitos. Los elementos v_{ij} toman dos posibles valores: 1 si el requisito interesa al cliente (tiene valor para él) y 0 si no está interesado en él.

Por otro lado, Del Sagrado et al. [11] definen la *satisfacción* asociada a un requisito, s_j, como la suma ponderada del valor que le dan los clientes, $s_j = \sum_{i=1}^{m} w_i * v_{ij}$. Según esta interpretación, no es necesario implementar todos los requisitos que interesan a un cliente para obtener una cierta satisfacción del mismo.

En cualquier caso, los requisitos presentan dependencias o interacciones entre ellos, imponiendo un orden de desarrollo determinado, lo que limita las alternativas para ser elegidos [3,8]. Las interacciones entre los requisitos representan restricciones al problema y se agrupan en dos tipos: *dependencias funcionales* o *estructurales* y *dependencias por recursos consumidos* [12]. En este trabajo nos centramos en las dependencias funcionales, que pueden definirse como:

- *Implicación o precedencia.* $r_i \Rightarrow r_j$. El requisito r_i no se puede seleccionar si el requisito r_j no ha sido ya implementado.
- *Combinación o acoplamiento.* $r_i \odot r_j$. Los requisitos r_i y r_j deben ser incluidos de forma conjunta en el software.
- *Exclusión.* $r_i \oplus r_j$. El requisito r_i no puede incluirse junto al requisito r_j.

El objetivo será encontrar $\hat{R} \subseteq R$ de forma que se maximice el valor a la vez que se minimiza el coste para el conjunto de requisitos seleccionados $\hat{R}$. El coste

viene dado por la función:

$$coste(\hat{R}) = \sum_{j,r_j \in \hat{R}}^{n} c_j, \tag{1}$$

mientras que el valor viene dado por las funciones:

$$valor(\hat{R}) = \sum_{i=1}^{m} w_i \prod_{j,r_j \in \hat{R}} v_{ij} \quad \text{y} \quad valor(\hat{R}) = \sum_{j,r_j \in \hat{R}}^{n} s_j, \tag{2}$$

en las interpretaciones de Xuan et al. [14] y Del Sagrado et al. [11], respectivamente. El problema es multi-objetivo y, por tanto no existe una solución única, sino un conjunto de soluciones Pareto óptimas, también llamadas *no dominadas* o *eficientes* [6].

3. Formulación del problema como ILP bi-objetivo

A partir de la descripción del problema, su formulación es bastante directa como programa lineal entero. Para ello, utilizaremos una variable binaria (puede tomar valores 0 y 1) por cada requisito seleccionable. Por simplicidad, aquí usaremos para dichas variables el mismo nombre que los requisitos asociados: r_1, r_2, etc. Para formar el programa lineal, es necesario transformar cada interacción funcional entre requisitos en una igualdad o desigualdad de expresiones lineales. Estas transformaciones se realizan de acuerdo al siguiente esquema:

- Implicación $r_i \Rightarrow r_j$: $r_i \leq r_j$.
- Combinación $r_i \odot r_j$: $r_i = r_j$.
- Exclusión $r_i \oplus r_j$: $r_i + r_j \leq 1$.

La función de coste es:

$$coste(r) = \sum_{j=1}^{n} c_j r_j. \tag{3}$$

Las funciones de valor en las interpretaciones de Xuan et al. y Del Sagrado et al. son:

$$valor(t) = \sum_{i=1}^{m} w_i t_i \quad \text{y} \quad valor(r) = \sum_{j=1}^{n} s_j r_j. \tag{4}$$

donde las variables t_i que aparecen en la función de valor de Xuan et al. indican si un cliente está o no satisfecho. La exigencia de que un cliente esté satisfecho sólo cuando todos sus requisitos están implementados en la interpretación de Xuan et al., implica la incorporación de restricciones de la forma $t_i \leq r_j$ si y sólo si $v_{ij} = 1$.

4. Algoritmos de resolución

En esta sección presentamos los distintos algoritmos basados en programación lineal entera utilizados para calcular el frente de Pareto del problema de selección de requisitos. Para la exposición de los algoritmos se considerará, sin pérdida de generalidad, que los dos objetivos deben ser minimizados. Llamaremos X al conjunto de soluciones que cumplen con todas las restricciones. Decimos que una solución $x \in X$ es *débilmente eficiente* cuando no existe $y \in X$ tal que $f_i(y) < f_i(x)$ para $i = 1, 2$, es decir, no existe solución que mejore a x en todos los objetivos. Un solución $y \in X$ *domina* a $x \in X$ si $f_i(y) \leq f_i(x)$ para $i = 1, 2$ y existe $j \in \{1, 2\}$ tal que $f_j(y) < f_j(x)$. Una solución $x \in X$ es *eficiente* si no existe $y \in X$ que la domine. Toda solución eficiente es débilmente eficiente [6]. Una solución $x \in X$ se denomina *solución soportada* si existe un vector de valores reales $\alpha_i \in \mathbb{R}$ tal que x también es solución del problema de minimización de $\sum_{i=1}^{2} \alpha_i f_i(x)$ sujeto a $x \in X$. Las principales contribuciones de este artículo son los algoritmos descritos en las secciones 4.4 y 4.5.

4.1. Algoritmo ε-constraint con un ILP por iteración

Este algoritmo se muestra en el Algoritmo 1 y es el utilizado en [13] para encontrar el frente de Pareto completo. Al principio calcula una solución z que minimice f_2 y la introduce en el conjunto FP, de soluciones óptimas de Pareto. A continuación asigna a ε el valor $f_1(z)-1$. Al entrar en el bucle, trata de minimizar f_2 sujeto a que el valor de f_1 sea menor que ε. El valor de f_1 en la nueva solución servirá para establecer el nuevo límite para f_1. El bucle termina cuando no existen soluciones con f_1 por debajo de ϵ, garantizándose de esta forma que no existirán más soluciones eficientes. El algoritmo resuelve un único subproblema en cada iteración (línea 5) y sólo puede garantizar obtener soluciones débilmente eficientes. Esto exige eliminar soluciones dominadas al finalizar el bucle (línea 9).

Algoritmo 1 ε-constraint con un ILP por iteración

```
1: z ← resolver {mín f2(x), sujeto a x ∈ X}
2: FP ← {z} // Frente de Pareto
3: ε ← f1(z) − 1
4: while ∃x ∈ X, f1(x) ≤ ε do
5:     z ← resolver {mín f2(x),  sujeto a  f1(x) ≤ ε, x ∈ X}
6:     FP = FP ∪{z}
7:     ε ← f1(z) − 1
8: end while
9: Eliminar de FP las soluciones dominadas
```

4.2. Algoritmo ε-constraint con dos ILPs por iteración

En el Algoritmo 2 mostramos una variante de ε-constraint que resuelve dos subproblemas por iteración en lugar de uno. El objetivo es encontrar en cada

iteración una solución eficiente, que se puede añadir a FP sin filtrar. El funcionamiento del bucle es el mismo que en el caso anterior. Este algoritmo fue propuesto para el NRP en [1], donde se usó un resolutor SAT en lugar de un resolutor ILP para resolver los problemas de optimización.

Podría parecer que el Algoritmo 1 es más rápido que el Algoritmo 2, ya que el esfuerzo computacional es menor, pero esta diferencia va disminuyendo a medida que el problema a resolver posea un conjunto mayor de soluciones débilmente eficientes. Supóngase el caso en que $|N| = k$ y $|wN| > 2k$, siendo N y wN los conjuntos de puntos eficientes (no dominados) y débilmente eficientes, respectivamente. El Algoritmo 1 tendrá que realizar en el bucle más de $2k$ iteraciones y el Algoritmo 2 sólo $2k$ iteraciones.

Algoritmo 2 ε-constraint con dos ILPs por iteración

```
1: z ← resolver {mín f2(x),  sujeto a  x ∈ X}
2: z ← resolver {mín f1(x),  sujeto a  f2(x) ≤ f2(z), x ∈ X}
3: FP ← {z} // Frente de Pareto
4: ε ← f1(z) − 1
5: while ∃x ∈ X, f1(x) ≤ ε do
6:     z ← resolver {mín f2(x),  sujeto a  f1(x) ≤ ε, x ∈ X}
7:     z ← resolver {mín f1(x),  sujeto a  f2(x) ≤ f2(z), x ∈ X}
8:     FP = FP ∪{z}
9:     ε ← f1(z) − 1
10: end while
```

4.3. Algoritmo ε-constraint aumentado (Aε-con)

El Algoritmo 3, conocido como *augmented ε-constraint* o AUGMECON, elimina las deficiencias de los Algoritmos 1 y 2. Por un lado, solo se resuelve un subproblema en cada iteración, y por otro, se garantiza que el punto no dominado obtenido es eficiente. El lector interesado puede consultar [9] para más detalles. En cada iteración, el algoritmo fija un valor positivo para la constante λ, que debe ser suficientemente pequeño para evitar que el algoritmo omita algunas de las soluciones eficientes, y lo bastante grande como para evitar problemas numéricos. En general, es suficiente tomar un valor de λ en $[10^{-3}, 10^{-6}]$ (véase [9]). En nuestro caso, hemos optado por calcular el valor de λ en cada iteración teniendo en cuenta el punto eficiente obtenido anteriormente. En particular la expresión que usamos es: $\lambda = 1/(f_1(z) - u_1)$, donde u_1 es una cota inferior de mín $f_1(x), x \in X$, es decir, la primera componente de un punto utópico.

Algoritmo 3 ε-constraint aumentado

1: $z \leftarrow$ resolver $\{\text{mín } f_2(x), \text{ sujeto a } x \in X\}$
2: $z \leftarrow$ resolver $\{\text{mín } f_1(x), \text{ sujeto a } f_2(x) \leq f_2(z), x \in X\}$
3: FP $\leftarrow \{z\}$ // Frente de Pareto
4: $\varepsilon \leftarrow f_1(z) - 1$
5: **while** $\exists x \in X, f_1(x) \leq \varepsilon$ **do**
6: Estimar un valor para $\lambda > 0$
7: $z \leftarrow$ resolver $\{\text{mín } f_2(x) - \lambda l, \text{ sujeto a } f_1(x) + l = \varepsilon, x \in X\}$
8: FP = FP $\cup\{z\}$
9: $\varepsilon \leftarrow f_1(z) - 1$
10: **end while**

4.4. Algoritmo *anytime* basado en *augmented weighted Tchebycheff* (AAWTcheby)

Todos los algoritmos anteriores encuentran las soluciones eficientes en orden lexicográfico de sus objetivos. El principal problema de ese orden se pone de manifiesto cuando se trata de resolver una instancia tan grande que requiere mucho tiempo de cómputo. En ese caso, los algoritmos encontrarán sólo un extremo del frente. Desde un punto de vista práctico al decisor le interesa tener un conjunto de soluciones eficientes que se encuentren lo mejor distribuidas posible en el espacio objetivo. Esto puede conseguirse diseñando algoritmos que "salten" en el espacio objetivo en busca de soluciones eficientes. Estas estrategias se conocen en inglés como *anytime*. Veerapen et al. [13] utilizaron una búsqueda dicotómica para lograr este objetivo. La búsqueda dicotómica, sin embargo, adolece de un grave problema: sólo es capaz de encontrar soluciones eficientes soportadas. Como consecuencia, en frentes cóncavos, la calidad del frente que calcula puede ser muy baja. En el presente trabajo proponemos dos técnicas *aytime* que son capaces de encontrar el frente completo con suficiente tiempo. Comenzamos en esta sección describiendo la primera de ellas.

El algoritmo aumentado y ponderado de Tchebycheff permite encontrar soluciones eficientes en una zona cualquiera del frente usando una sola ejecución de resolutor ILP. La zona a explorar viene determinada por un par de puntos (generalmente eficientes) $(z^{(1)}, z^{(2)})$, que asumimos ordenados de tal forma que $z_1^{(1)} < z_1^{(2)}$. A partir de estos puntos, el algoritmo resuelve el siguiente problema lineal en cada iteración:

$$\text{mín } y \tag{5}$$

sujeto a

$$y \geq -(\zeta_1 - \zeta_0)(f_1(x) - \xi_1) + (\xi_1 - \xi_0)(f_2(x) - \zeta_1) \tag{6}$$
$$y \geq -(\zeta_2 - \zeta_1)(f_1(x) - \xi_1) + (\xi_2 - \xi_1)(f_2(x) - \zeta_1) \tag{7}$$
$$f_1(x) \leq \xi_2 \tag{8}$$
$$f_2(x) \leq \zeta_0 \tag{9}$$

donde los valores de ζ_i y ξ_i son: $\xi_0 = z_1^{(1)}$, $\zeta_0 = z_2^{(1)}$, $\xi_1 = z_1^{(2)} - 1/2$, $\zeta_1 = z_2^{(1)} - 1/2$, $\xi_2 = z_1^{(2)}$, $\zeta_2 = z_2^{(2)}$. El problema anterior devuelve una solución eficiente que se encuentra entre $z^{(1)}$ y $z^{(2)}$. Para más detalles consultar [4].

A partir de este programa podemos construir un algoritmo que primero calcula los dos óptimos lexicográficos (líneas 1 y 2) y, a continuación, explora el espacio entre ellos. Cada vez que encuentra un nuevo punto eficiente entre dos existentes, lo añade al frente de Pareto y divide el rectángulo explorado en dos para explorarlos más adelante (línea 10). Cuando ya no quedan más rectángulos sin explorar el algoritmo termina. En nuestra implementación los rectángulos son explorados por orden de área, el de mayor área se explora primero. Esto permite obtener un punto en cada iteración con potencial para maximizar el hipervolumen cubierto hasta ese momento.

Algoritmo 4 *Anytime augmented weighted* Tchebycheff

```
1: z^(1) ← calcular óptimo lexicográfico para el orden (f1, f2)
2: z^(2) ← calcular óptimo lexicográfico para el orden (f2, f1)
3: FP ← {z^(1), z^(2)} // Frente de Pareto
4: Cola ← {(z^(1), z^(2))}
5: while Cola ≠ ∅ do
6:    (z^(1), z^(2)) ← extraerParDeMayorArea(Cola)
7:    z ← resolverTchebycheff((z^(1), z^(2)))
8:    if z no dominado en (z^(1), z^(2)) then
9:       FP = FP ∪{z}
10:      Cola ← Cola ∪{(z^(1), z), (z, z^(2))}
11:   end if
12: end while
```

4.5. Algoritmo *anytime* basado en ε-constraint aumentado (AAε-con)

El Algoritmo 5 actúa de forma similar al anterior, analizando el rectángulo en el espacio objetivo delimitado por pares de puntos. La diferencia fundamental entre ambos algoritmos es que en AAε-con se utiliza el algoritmo ε-constraint aumentado para encontrar la solución eficiente dentro de ese rectángulo.

El algoritmo ε-constraint aumentado necesita un valor de ε, que establecemos al punto medio entre $z^{(1)}$ y $z^{(2)}$ (línea 7). Si existe una solución eficiente con f_1 menor que dicho ε, el algoritmo la encontrará y la incorporará a FP, salvo que sea una solución dominada (es decir, que la haya encontrado antes). Si no encuentra ninguna solución, entonces debe añadir a la cola de exploración la mitad del rectángulo que no ha sido explorado, es decir, aquella con f_1 mayor que ε.

Algoritmo 5 *Anytime* ε-constraint aumentado

1: $z^{(1)} \leftarrow$ calcular óptimo lexicográfico para el orden (f_1, f_2)
2: $z^{(2)} \leftarrow$ calcular óptimo lexicográfico para el orden (f_2, f_1)
3: FP $\leftarrow \{z^{(1)}, z^{(2)}\}$ // Frente de Pareto
4: Cola $\leftarrow \{(z^{(1)}, z^{(2)})\}$
5: **while** Cola $\neq \emptyset$ **do**
6: $(z^{(1)}, z^{(2)}) \leftarrow$ extraerParDeMayorArea(Cola)
7: $\varepsilon \leftarrow (z_1^{(1)} + z_1^{(2)})/2$
8: $z \leftarrow$ resolver $\{\text{mín } f_2(x) - \lambda l, \quad s.a. \quad f_1(x) + l = \varepsilon, x \in X\}$
9: **if** z no dominado en $(z^{(1)}, z^{(2)})$ **then**
10: FP = FP $\cup\{z\}$
11: Cola $\leftarrow$ Cola $\cup\{(z^{(1)}, z), (z, z^{(2)})\}$
12: **else**
13: Cola $\leftarrow$ Cola $\cup\{((\varepsilon, z_2^{(1)}), z^{(2)})\}$
14: **end if**
15: **end while**

5. Estudio experimental

En esta sección comparamos los resultados obtenidos por los distintos algoritmos de programación lineal entera. En particular, estamos interesados en estudiar la calidad del frente de Pareto cuando limitamos el tiempo de ejecución de los algoritmos. Este escenario puede corresponderse con el habitual en la práctica, en especial, cuando se usan metodologías ágiles y debe resolverse el problema de determinar los requisitos a implementar en la siguiente iteración. Por otro lado, nos gustaría comparar los resultados de programación lineal entera con los algoritmos metaheurísticos que se han venido utilizando para resolver este problema hasta el momento. El código con la implementación de los algoritmos utilizados en el estudio está publicado en la URL `https://github.com/jfrchicanog/NextReleaseProblem`. En dicha URL también puede encontrarse un enlace a las instancias.

5.1. Instancias

Utilizaremos dos conjuntos de instancias para los experimentos. Por un lado, usaremos las 17 instancias de Xuan et al. [14], utilizadas también en el trabajo de Veerapen et al. [13]. Este conjunto a su vez contiene cinco instancias clásicas y 12 realistas. Tienen entre 140 y 4368 requisitos y la única posible relación entre requisitos es la implicación. El segundo conjunto de instancias está formado por dos instancias que han sido previamente utilizadas en el trabajo de Del Sagrado et al. [11]. Las instancias tienen 20 y 100 requisitos, respectivamente, y se caracterizan porque, además de la implicación, aparece la relación de simultaneidad. En la primer columna de la tabla 1 aparece el número de requisitos de cada instancia junto a su nombre.

5.2. Cálculo del frente con límite de tiempo

El trabajo de Veerapen et al. [13] muestra que, aunque los resolutores de programación lineal entera han avanzado hasta el punto de resultar de utilidad para el problema de la selección de requisitos, los tiempos de ejecución crecen rápidamente con el tamaño de las instancias. Así, de las 19 instancias utilizadas en dicho trabajo, sólo en tres los algoritmos terminan de calcular el frente de Pareto en menos de 30 segundos (las instancias que resultan ser más rápidas de resolver son las dos reales). Por eso, en este estudio nos planteamos la siguiente pregunta de investigación: **RQ1** ¿cuál es la calidad de la parte del frente de Pareto encontrada por cada algoritmo cuando limitamos el tiempo de ejecución de los algoritmos?

A priori, pensamos que los algoritmos *anytime* deben ser mejores que aquellos que buscan las soluciones del frente en orden lexicográfico cuando el límite de tiempo es bajo en comparación con el tiempo requerido para encontrar todo el frente. Esperamos, no obstante, que los algoritmos que siguen el orden lexicográfico aventajen a los *anytime* cuando el límite de tiempo se acerca al requerido para encontrar todas las soluciones.

Hemos limitado el tiempo que los algoritmos se ejecutan a 300 segundos y hemos realizado una sola ejecución de los algoritmos, ya que son deterministas. El tiempo de ejecución de los algoritmos puede depender de la carga del sistema operativo. No obstante, este error en el tiempo es muy inferior al límite de tiempo, por lo que no vemos necesaria la realización de más de una ejecución de los algoritmos. La comparación de los distintos algoritmos la haremos calculando el hipervolumen basado en el punto de Nadir[4] obtenido por el frente generado por cada algoritmo al finalizar el límite de tiempo. Los resultados se muestran en la tabla 1. En algunas instancias (en particular, `nrp1`, `dataset1` y `dataset2`), los algoritmos terminan en menos de 300 segundos y, como consecuencia, todos los algoritmos alcanzan el máximo hipervolumen.

Podemos observar que, con la única excepción de `nrp5`, los algoritmos que consiguen un mayor hipervolumen son los algoritmos *anytime*. Por tanto, vemos que cumplen el objetivo para el que fueron diseñados. Como habíamos adelantado, cuando el tiempo de ejecución de los algoritmos que siguen un orden lexicográfico es cercano o menor que el límite de tiempo, son estos algoritmos los que resultan más eficientes, ya que son más rápidos encontrando el frente completo. Este es el caso de `nrp5` que, según [13], es la instancia de Xuan et al. más rápida de resolver después de `nrp1`.

Entre los algoritmos *anytime*, no podemos identificar un claro ganador. Vemos que AAWTcheby consigue resultados ligeramente mejores en algunas instancias mientras que AAε-con los mejora en otras, pero las diferencias son pequeñas.

[4] El hipervolumen en este caso es el área comprendida entre el punto de Nadir y las soluciones del frente obtenido.

Tabla 1. Hipervolumen de las soluciones del frente obtenidas por los distintos algoritmos en 300 segundos. Se destacan con fondo oscuro las celdas con valores máximos para cada instancia.

	Orden lexicográfico			Anytime	
Instancia (req.)	**ε-con 1-ILP**	**ε-con 2-ILPs**	**Aε-con**	**AAWTcheby**	**AAε-con**
dataset1 (20)	$5{,}23{\cdot}10^{4}$	$5{,}23{\cdot}10^{4}$	$5{,}23{\cdot}10^{4}$	$5{,}23{\cdot}10^{4}$	$5{,}23{\cdot}10^{4}$
dataset2 (100)	$1{,}80{\cdot}10^{6}$	$1{,}80{\cdot}10^{6}$	$1{,}80{\cdot}10^{6}$	$1{,}80{\cdot}10^{6}$	$1{,}80{\cdot}10^{6}$
nrp1 (140)	$1{,}32{\cdot}10^{6}$	$1{,}32{\cdot}10^{6}$	$1{,}32{\cdot}10^{6}$	$1{,}32{\cdot}10^{6}$	$1{,}32{\cdot}10^{6}$
nrp2 (620)	$1{,}75{\cdot}10^{6}$	$1{,}06{\cdot}10^{6}$	$1{,}57{\cdot}10^{6}$	$3{,}65{\cdot}10^{7}$	$3{,}68{\cdot}10^{7}$
nrp3 (1500)	$3{,}04{\cdot}10^{6}$	$1{,}15{\cdot}10^{6}$	$9{,}88{\cdot}10^{5}$	$5{,}84{\cdot}10^{7}$	$5{,}85{\cdot}10^{7}$
nrp4 (3250)	$1{,}21{\cdot}10^{6}$	$6{,}98{\cdot}10^{5}$	$1{,}29{\cdot}10^{6}$	$2{,}07{\cdot}10^{8}$	$2{,}16{\cdot}10^{8}$
nrp5 (1500)	$5{,}60{\cdot}10^{7}$	$1{,}19{\cdot}10^{7}$	$3{,}00{\cdot}10^{7}$	$5{,}25{\cdot}10^{7}$	$5{,}60{\cdot}10^{7}$
nrp-e1 (3502)	$3{,}14{\cdot}10^{6}$	$1{,}74{\cdot}10^{6}$	$4{,}58{\cdot}10^{5}$	$1{,}34{\cdot}10^{8}$	$1{,}34{\cdot}10^{8}$
nrp-e2 (4254)	$2{,}69{\cdot}10^{6}$	$1{,}05{\cdot}10^{6}$	$2{,}31{\cdot}10^{6}$	$1{,}52{\cdot}10^{8}$	$1{,}52{\cdot}10^{8}$
nrp-e3 (2844)	$6{,}84{\cdot}10^{6}$	$2{,}87{\cdot}10^{6}$	$5{,}42{\cdot}10^{5}$	$8{,}94{\cdot}10^{7}$	$8{,}94{\cdot}10^{7}$
nrp-e4 (3186)	$4{,}10{\cdot}10^{6}$	$2{,}08{\cdot}10^{6}$	$5{,}20{\cdot}10^{5}$	$8{,}82{\cdot}10^{7}$	$8{,}81{\cdot}10^{7}$
nrp-g1 (2690)	$1{,}26{\cdot}10^{7}$	$4{,}77{\cdot}10^{6}$	$7{,}82{\cdot}10^{6}$	$1{,}09{\cdot}10^{8}$	$1{,}09{\cdot}10^{8}$
nrp-g2 (2650)	$1{,}15{\cdot}10^{7}$	$3{,}58{\cdot}10^{6}$	$8{,}38{\cdot}10^{6}$	$7{,}56{\cdot}10^{7}$	$7{,}56{\cdot}10^{7}$
nrp-g3 (2512)	$2{,}01{\cdot}10^{7}$	$9{,}39{\cdot}10^{6}$	$1{,}79{\cdot}10^{6}$	$9{,}64{\cdot}10^{7}$	$9{,}63{\cdot}10^{7}$
nrp-g4 (2246)	$1{,}94{\cdot}10^{7}$	$5{,}13{\cdot}10^{6}$	$1{,}24{\cdot}10^{7}$	$5{,}96{\cdot}10^{7}$	$5{,}97{\cdot}10^{7}$
nrp-m1 (4060)	$1{,}86{\cdot}10^{6}$	$8{,}59{\cdot}10^{5}$	$4{,}20{\cdot}10^{5}$	$2{,}24{\cdot}10^{8}$	$2{,}24{\cdot}10^{8}$
nrp-m2 (4368)	$1{,}51{\cdot}10^{6}$	$9{,}34{\cdot}10^{5}$	$3{,}98{\cdot}10^{5}$	$1{,}96{\cdot}10^{8}$	$1{,}95{\cdot}10^{8}$
nrp-m3 (3566)	$2{,}06{\cdot}10^{6}$	$8{,}95{\cdot}10^{5}$	$3{,}72{\cdot}10^{5}$	$1{,}92{\cdot}10^{8}$	$1{,}92{\cdot}10^{8}$
nrp-m4 (3643)	$2{,}34{\cdot}10^{6}$	$1{,}20{\cdot}10^{6}$	$5{,}08{\cdot}10^{5}$	$1{,}48{\cdot}10^{8}$	$1{,}48{\cdot}10^{8}$

5.3. Comparación con metaheurísticas

El problema de selección de requisitos bi-objetivo se ha resuelto en trabajos previos usando técnicas metaheurísticas, como NSGA-II, ACO y GRASP [5,11,15]. Tanto este trabajo como [13], muestran que los algoritmos exactos basados en programación lineal entera son muy buenos encontrando un frente de alta calidad, en ocasiones en un tiempo razonable. Por tanto, nos planteamos la siguiente pregunta de investigación: **RQ2** ¿cuándo conviene utilizar metaheurísticas para resolver el problema y cuándo es mejor usar algoritmos basados en programación lineal entera?

De nuevo utilizaremos aquí las 17 instancias de Xuan et al. y las 2 de Del Sagrado. Para el caso de las 17 instancias de Xuan, Veerapen et al. ya respondieron a dicha pregunta en [13]. Allí compararon el método ε-constraint y la búsqueda dicotómica *anytime* (ADS) con NSGA-II. Su conclusión fue que para la mayoría de las instancias y, en particular, las de mayor tamaño, ADS obtenía un hipervolumen más alto que NSGA-II en aproximadamente el mismo tiempo (unos 90 segundos). Por otro lado, NSGA-II aventajaba a ADS en instancias muy pequeñas, que podían resolverse de manera completa usando ε-constraint. La única instancia para la que NSGA-II obtuvo un frente con mejor hipervolumen que una técnica basada en ILP en un tiempo razonable fue `nrp2`.

Para completar estos resultados, hemos ejecutado los dos algoritmos *anytime* durante el mismo tiempo que Veerapen et al. ejecutaron NSGA-II (el tiempo requerido para realizar 200 iteraciones del bucle principal de NSGA-II). Los resultados se observan en la tabla 2. La máquina empleada en [13] fue una Intel Core i7-3770 con cuatro núcleos a 3.4 GHz, que es aproximadamente entre 1,5 y

Tabla 2. Hipervolumen obtenido por NSGA-II y los dos algoritmos *anytime* en el tiempo que NSGA-II necesita para realizar 200 iteraciones (indicado en la segunda columna) [13]. Se destacan las celdas conteniendo valores más altos por instancia.

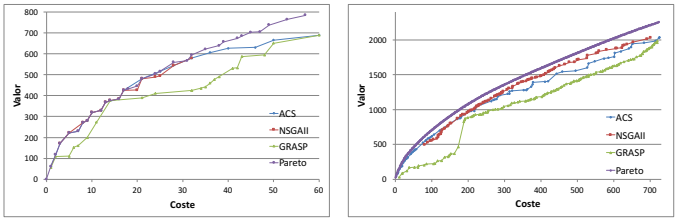
Instancia	Tiempo (s)	NSGA-II	AAWTcheby	AAε-con
nrp1	53	$1,26 \cdot 10^6$	$1,32 \cdot 10^6$	$1,32 \cdot 10^6$
nrp2	64	$2,74 \cdot 10^7$	$3,50 \cdot 10^7$	$3,62 \cdot 10^7$
nrp3	69	$4,74 \cdot 10^7$	$5,78 \cdot 10^7$	$5,82 \cdot 10^7$
nrp4	71	$1,59 \cdot 10^8$	$1,95 \cdot 10^8$	$2,13 \cdot 10^8$
nrp5	66	$3,97 \cdot 10^7$	$5,12 \cdot 10^7$	$5,60 \cdot 10^7$
nrp-e1	79	$3,97 \cdot 10^8$	$1,34 \cdot 10^8$	$1,33 \cdot 10^8$
nrp-e2	76	$1,20 \cdot 10^8$	$1,51 \cdot 10^8$	$1,52 \cdot 10^8$
nrp-e3	76	$8,07 \cdot 10^7$	$8,93 \cdot 10^7$	$8,90 \cdot 10^7$
nrp-e4	77	$7,98 \cdot 10^7$	$8,79 \cdot 10^7$	$8,74 \cdot 10^7$
nrp-g1	79	$0,98 \cdot 10^8$	$1,09 \cdot 10^8$	$1,09 \cdot 10^8$
nrp-g2	76	$6,92 \cdot 10^7$	$7,55 \cdot 10^7$	$7,55 \cdot 10^7$
nrp-g3	79	$8,78 \cdot 10^7$	$9,62 \cdot 10^7$	$9,60 \cdot 10^7$
nrp-g4	77	$5,51 \cdot 10^7$	$5,96 \cdot 10^7$	$5,96 \cdot 10^7$
nrp-m1	81	$1,96 \cdot 10^8$	$2,23 \cdot 10^8$	$2,23 \cdot 10^8$
nrp-m2	78	$1,74 \cdot 10^8$	$1,94 \cdot 10^8$	$1,94 \cdot 10^8$
nrp-m3	82	$1,67 \cdot 10^8$	$1,91 \cdot 10^8$	$1,91 \cdot 10^8$
nrp-m4	79	$1,32 \cdot 10^8$	$1,47 \cdot 10^8$	$1,47 \cdot 10^8$

2 veces más rápida que la usada en nuestros experimentos. Aún así, se observa que los algoritmos *anytime* aquí propuestos son mejores que NSGA-II.

Con respecto a las instancias de Del Sagrado et al., observamos en la sección anterior que son instancias cuya solución exacta se puede calcular en un tiempo muy corto (menos de 30 segundos), lo que significa que, en la práctica, basta con aplicar una técnica exacta para resolverlos. En cualquier caso, en la tabla 3 mostramos el hipervolumen y el tiempo de ejecución requerido por ACO, GRASP y NSGA-II para resolver estas instancias, y en la Figura 1 mostramos el frente de Pareto y las aproximaciones obtenidas por las metaheurísticas.

Tabla 3. Comparación del hipervolumen obtenido y tiempo requerido por NSGA-II, GRASP y ACS en las instancias de Del Sagrado et al. En el caso de las metaheurísticas el tiempo e hipervolumen son promedios de 100 ejecuciones independientes.

		dataset1		dataset2	
Algoritmo	**% esfuerzo**	**Hipervolumen**	**Tiempo (ms)**	**Hipervolumen**	**Tiempo (ms)**
NSGA-II	30	6843	1892	218138	28128
	50	15677	1981	495949	35046
	70	24409	2034	873384	38300
GRASP	30	5851	363	112419	28920
	50	14508	1208	425642	118340
	70	24474	840	769613	324604
ACS	30	7805	639	234583	616874
	50	18153	788	527685	770221
	70	29196	837	902769	881951
ε-con. 1-ILP	100	52271	290	1797324	11553



(a) dataset1 (b) dataset2

Figura 1. Frente de Pareto y aproximaciones de los algoritmos metaheurísticos.

Hemos de indicar que estos tiempos se refieren de nuevo a una máquina diferente (Pentium 4 a 3,2 GHz) y el objetivo no era encontrar el frente completo, sino sólo una parte limitada por el 70 % de la suma de esfuerzos. Aún así, se puede observar que los algoritmos requieren un mayor tiempo de ejecución y, en el caso de `dataset2`, los frentes aproximados obtenidos por las metaheurísticas se encuentran lejos del frente de Pareto.

En resumen, podemos concluir que todas las instancias utilizadas en este trabajo se pueden resolver satisfactoriamente empleando técnicas basadas en programación lineal entera como AAε-con, si la instancia es grande, o ε-constraint, si es pequeña.

6. Conclusiones y trabajo futuro

En este trabajo hemos propuesto dos algoritmos basados en programación lineal entera para encontrar un conjunto de soluciones eficientes bien distribuidas en el espacio objetivo en cualquier momento de la búsqueda. Estos algoritmos, con el tiempo suficiente son capaces de encontrar el frente completo. Hemos comparado los algoritmos con otros diseñados para encontrar el frente completo y observamos una clara ventaja en el hipervolumen al detener todos los algoritmos tras un límite de tiempo. También hemos comparado los algoritmos basados en ILP con técnicas metaheurísticas y deducimos que los primeros son más rápidos y obtienen mejores soluciones.

Como trabajo futuro se puede estudiar el impacto en la formulación y en las técnicas basadas en ILP que tendría considerar la existencia de incertidumbre en el coste de los requisitos. También sería interesante explorar con más detalle la razón por la que los distintos algoritmos basados en ILP tienen el rendimiento tan dispar que presentan. Esto puede permitir desarrollar nuevos y mejores algoritmos para este problema.

Agradecimientos

El trabajo ha sido parcialmente financiado por la Universidad de Málaga, Andalucía Tech y el Ministerio de Economía y Competitividad mediante la red TIN2015-71841-REDT y el proyecto TIN2014-57341-R.

Referencias

1. del Águila, I., del Sagrado, J., Chicano, F., Alba, E.: Resolviendo un problema multi-objetivo de selección de requisitos mediante resolutores del problema SAT. In: XX Jornadas de Ingeniería del Software y Bases de Datos. SISTEDES, Santander, España (2015)
2. Berander, P., Svahnberg, M.: Evaluating two ways of calculating priorities in requirements hierarchies–an experiment on hierarchical cumulative voting. Journal of Systems and Software 82(5), 836–850 (2009)
3. Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., och Dag, J.N.: An industrial survey of requirements interdependencies in software product release planning. In: Proceedings of IEEE RE. pp. 84–93. IEEE Computer Society (2001)
4. Dächert, K., Gorski, J., Klamroth, K.: An augmented weighted tchebycheff method with adaptively chosen parameters for discrete bicriteria optimization problems. Computers & Operations Research 39(12), 2929 – 2943 (2012)
5. Durillo, J.J., Zhang, Y., Alba, E., Harman, M., Nebro, A.J.: A study of the bi-objective next release problem. Empirical Software Engineering 16(1), 29–60 (2011)
6. Ehrgott, M.: Multicriteria optimization. Springer (2005)
7. IIBA, A.: guide to the business analysis body of knowledge (babok guide). International Institute of Business Analysis (IIBA) (2009)
8. Karlsson, J., Olsson, S., Ryan, K.: Improving practical support for large-scale requirement prioritising. Requirement Engineering 2(1), 51–60 (1997)
9. Mavrotas, G.: Effective implementation of the ε-constraint method in multi-objective mathematical programming problems. Applied Mathematics and Computation 213(2), 455 – 465 (2009)
10. Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering. pp. 35–46. ACM (2000)
11. del Sagrado, J., del Águila, I.M., Orellana, F.J.: Multi-objective ant colony optimization for requirements selection. Empirical Software Engineering 20(3), 577–610 (2015)
12. del Sagrado, J., del Águila, I.M., Orellana, F.J.: Requirements interaction in the next release problem. In: Proceedings of 13th Annual Genetic and Evolutionary Computation Conference (GECCO 2011), Dublin, Ireland. pp. 241–242 (2011)
13. Veerapen, N., Ochoa, G., Harman, M., Burke, E.K.: An integer linear programming approach to the single and bi-objective next release problem. Information and Software Technology 65(0), 1 – 13 (2015)
14. Xuan, J., Jiang, H., Ren, Z., Luo, Z.: Solving the large scale next release problem with a backbone-based multilevel algorithm. Software Engineering, IEEE Transactions on 38(5), 1195–1212 (2012)
15. Zhang, Y., Harman, M., Mansouri, S.A.: The multi-objective next release problem. In: In Proceedings of GECCO. pp. 1129–1137 (2007)

Aplicando programación lineal entera a la búsqueda de conjuntos de productos de prueba priorizados para líneas de productos software

Javier Ferrer[1], Francisco Chicano[1],
Roberto E. Lopez-Herrejon[2] y Enrique Alba[1]

[1] Universidad de Málaga, Málaga, Spain
{ferrer, chicano, eat}@lcc.uma.es
[2] Systems Engineering and Automation
Johannes Kepler University, Linz, Austria
roberto.lopez@jku.at

Resumen Las líneas de productos software son familias de productos que están íntimamente relacionados entre sí, normalmente formados por combinaciones de un conjunto de características software. Generalmente no es factible testar todos los productos de la familia, ya que el número de productos es muy elevado debido a la explosión combinatoria de características. Por este motivo, se han propuesto criterios de cobertura que pretenden probar al menos todas las interacciones entre características sin necesidad de probar todos los productos, por ejemplo todos los pares de características (*pairwise coverage*). Además, es deseable testar primero los productos compuestos por un conjunto de características prioritarias. Este problema es conocido como *Prioritized Pairwise Test Data Generation*. En este trabajo proponemos una técnica basada en programación lineal entera para generar este conjunto de pruebas priorizado. Nuestro estudio revela que la propuesta basada en programación lineal entera consigue mejores resultados estadísticamente tanto en calidad como en tiempo de computación con respecto a las técnicas existentes para este problema.

Palabras clave Pruebas de interacción combinatoria, líneas de productos software, prioridades, programación lineal entera

1. Introducción

Las líneas de productos software (*SPL*) son familias de sistemas software relacionados, las cuales poseen diferentes combinaciones de características [1]. La gestión efectiva de la variabilidad es crucial para obtener beneficios de las SPLs como el incremento en la reutilización, la personalización rápida, y la reducción del tiempo de llegada al mercado. Debido al gran número de combinaciones de características que es típico en las SPLs, los modelos con alta variabilidad suponen un desafío para el campo de las pruebas de programas. Recientemente se

han propuesto muchos enfoques de pruebas [2–5]), sin embargo, todavía hay potencial de mejora ya que la mayoría de enfoques ya propuestos son aproximados y no suelen conseguir la solución óptima. Por contra, nosotros proponemos un enfoque basado en Programación Lineal Entera (en inglés *Integer Linear Programming* - ILP) para la generación del conjunto mínimo de productos de prueba en SPLs. Aunque la resolución de programas lineales enteros es un problema NP-difícil en general, con un coste computacional exponencial en el peor caso, los resolutores actuales, como CPLEX[3] o Gurobi[4], incluyen sofisticadas estrategias de búsqueda que les permiten resolver una gran cantidad de instancias de ILP en pocos segundos. Hasta donde llega nuestro conocimiento, esta técnica no se ha aplicado anteriormente a este problema.

En este artículo presentamos un algoritmo Ávido basado en Programación Lineal Entera (APLE) que genera un conjunto priorizado de pruebas para SPLs usando el criterio de cobertura *pairwise*, en el cual se deben cubrir todos los pares de características existentes en la SPL. APLE recibe como entrada un modelo de características (en inglés *Feature Model* - FM) y un conjunto de productos ponderados. Su objetivo es generar un conjunto de productos que cubren los pares de características deseados siguiendo diferentes esquemas de asignación de prioridades, con los que se van a generar diferentes ponderaciones para las características. Este esquema ha sido propuesto en [3] y ha sido recientemente aplicado de forma satisfactoria en la industria.

En nuestra experimentación validamos nuestra propuesta frente a dos algoritmos del estado del arte, un algoritmo ávido que genera soluciones competitivas en poco tiempo, llamado prioritized-ICPL (pICPL) [3] y un algoritmo genético llamado *Prioritized Pairwise Genetic Solver* (PPGS) [6] que obtiene soluciones de mejor calidad que pICPL pero generalmente usando un tiempo mayor. Nuestra comparativa abarca un total de 235 FMs con un amplio rango de características y productos, usando tres métodos de asignación de prioridades a los productos y cinco estrategias de selección de productos.

Nuestro estudio ha revelado que APLE obtiene conjuntos de pruebas priorizadas más pequeños en un tiempo menor para diferentes porcentajes de cobertura ponderada, con respecto a los resultados obtenidos con PPGS y pICPL. Estos resultados muestran que el uso de técnicas exactas en combinación con algoritmos ávidos constituye una estrategia muy efectiva para la generación de conjuntos de productos de prueba para SPLs. Nuestras principales contribuciones en este artículo son las siguientes:

- Propuesta de un algoritmo constructivo ávido basado en ILP.
- Evaluación del rendimiento de APLE en comparación con PPGS e pICPL.

El resto del artículo se organiza de la siguiente manera. En la Sección 2 presentamos los modelos de características. En la Sección 3 se formaliza el problema de la generación de conjuntos de productos de prueba priorizados en SPL. La Sección 4 describe nuestra propuesta algorítmica. En la Sección 5 presentamos los demás algoritmos objeto de la comparación, los métodos de asignación de

[3] `http://www-03.ibm.com/software/products/es/ibmilogcpleoptistud`
[4] `https://www.gurobi.com`

prioridades y las instancias usadas en la experimentación. La Sección 6 está dedicada al análisis estadístico de los resultados y la Sección 7 a describir las amenazas a la validez del estudio. Por último, en la Sección 8 ofrecemos las conclusiones obtenidas y los siguientes pasos de nuestra investigación.

2. Modelos de Características

Los modelos de características son un estándar *de facto* para modelar las características comunes y variables de un sistema (representadas por cajas etiquetadas) y sus relaciones (representadas con líneas) formando una estructura de tipo árbol, especificando así un conjunto de combinaciones de características que dan lugar a múltiples configuraciones diferentes [7]. Cada característica distinta de la raíz tiene una sola característica padre y puede tener un conjunto de características hijas. Nótese que una característica hija sólo puede ser incluida en una configuración si y sólo si, su padre es incluido también. Para ilustrar estos conceptos vamos a usar un FM (Figura 1) extraído del repositorio SPLOT [8].

Hay cuatro tipos de relaciones jerárquicas entre características:

- *Características opcionales:* son representadas con un círculo vacío e indica que esta característica puede o no ser seleccionada si su padre es seleccionado. En nuestra instancia de ejemplo sería la característica `Engine`.
- *Características obligatorias:* son representadas con un círculo relleno, y deben ser seleccionadas si su padre es seleccionado. En nuestra instancia de ejemplo serían obligatorias: `Wing` y `Materials`.
- *Relaciones Or-inclusivas:* son representadas como arcos independientes rellenos que abarcan un conjunto de líneas que conectan la característica padre con las características hijas. Indican que al menos una característica debe ser seleccionada si el padre es seleccionado. En nuestra instancia de ejemplo, la relación de `Wing` o `Materials` con sus hijos es de este tipo.
- *Relaciones Or-exclusivas:* son representadas como arcos independientes vacíos que abarcan un conjunto de líneas que conectan la característica padre con las características hijas. Indican que exactamente una característica debe ser seleccionada si el padre es seleccionado. En nuestra instancia de ejemplo la relaciones de `Engine` con sus respectivos hijos son Or-exclusivas.

Además de las relaciones padre-hijo, las características se pueden relacionar también con otras ramas del modelo de características, son las restricciones conocidas como *Cross-Tree Constraints (CTC)* [9]. Estas restricciones, así como las impuestas por las relaciones jerárquicas entre características, son expresadas y comprobadas usando lógica proposicional (para más detalles consultar [9]).

3. Formalización del Problema: *Prioritized Pairwise Test Data Generation*

En esta sección proporcionamos una descripción formal del problema de la generación de conjunto de pruebas priorizados por pares y del esquema de prioridades implementado por las propuestas algorítmicas estudiadas en este trabajo.

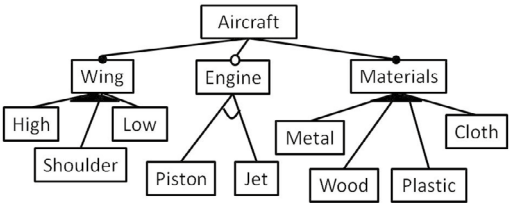


Figura 1. Modelo de características de Aircraft

Definición 1. *Lista de Características (LC) es la lista de característica del modelo de características.*

Definición 2. *Conjunto de Características (CC) es un par* $(sel, \overline{sel})$ *donde sel y* $\overline{sel}$ *son respectivamente los conjuntos de características seleccionadas y no seleccionadas de un producto. Sea LC una lista de características, entonces sel,* $\overline{sel} \subseteq LC$, $sel \cap \overline{sel} = \emptyset$, *y* $sel \cup \overline{sel} = LC$. *Si p es un producto, los términos* $p.sel$ *y* $p.\overline{sel}$ *son el conjunto de características seleccionadas y no seleccionadas del producto p, respectivamente.*

Definición 3. *Un conjunto de características cc es válido en un modelo de características* fm, *es decir,* $valido(cc, fm)$ *es verdadero, si y solo si cc no contradice ninguna restricción introducida por* fm.

Las pruebas de interacción combinatorias son un enfoque que construye conjuntos de pruebas (*test suites*) que nos llevan a probar de forma sistemática todas las configuraciones de un sistema [10]. En este trabajo vamos a utilizar el enfoque *pairwise*, que es el más utilizado en pruebas combinatorias y está basado en la suposición de que la mayoría de errores originados en un parámetro es causado por la interacción de dos valores [11]. Este criterio se satisface si para cada par de características f_1 y f_2, podemos encontrar cuatro productos en el conjunto generado que contengan las cuatro posibles combinaciones de presencia/ausencia de dichas características: $(f_1, f_2), (f_1, \overline{f_2}), (\overline{f_1}, f_2)$ y $(\overline{f_1}, \overline{f_2})$. Cuando se aplica esta técnica a FMs, la idea es generar un conjunto de productos válidos, donde los errores posibles se manifiestan con alta probabilidad, sin tener que probar de forma exhaustiva todas las posibles configuraciones. Además, gracias a la priorización, vamos a testar en primer lugar los productos con aquellas características que son más frecuentes o importantes en nuestros sistemas.

Definición 4. *Un producto priorizado pp es un par* (cc, w), *donde cc representa un conjunto de características válido en un modelo e características* fm *y* $w \in \mathbb{R}$ *representa su peso. Sean* pp_i *y* pp_j *dos productos priorizados. Decimos que* pp_i *tiene prioridad más alta que* pp_j *cuando el peso de* pp_i *es más alto que el peso de* pp_j, *es decir,* $pp_i.w > pp_j.w$.

Definición 5. *Una configuración por pares pc es un par* $(sel, \overline{sel})$ *que representa un producto parcialmente configurado, definidos por la selección de dos características de LC, es decir,* $pc.sel \cup pc.\overline{sel} \subseteq LC$, $pc.sel \cap pc.\overline{sel} = \emptyset$ *y* $|pc.sel \cup pc.\overline{sel}| = 2$. *Decimos que pc se cubre con un conjunto de características cc cuando* $pc.sel \subseteq cc.sel$ *y* $pc.\overline{sel} \subseteq cc.\overline{sel}$.

Definición 6. *Una configuración por pares ponderada wpc es un par* (pc, w) *donde pc es una configuración por pares y* $w \in \mathbb{R}$ *representa su peso calculado de la siguiente manera. Sea* PP *un conjunto de productos priorizados y* $PP_{pc} \subseteq PP$, *tal que* PP_{pc} *contiene todos los productos priorizados de* PP *que cubren wpc.pc, es decir,* $PP_{pc} = \{pp \in PP | pp.cc \text{ cubre } wpc.pc\}$. *Entonces* $w = \sum_{p \in PP_{pc}} p.w$.

Dada una colección de conjuntos de características $ppCA$ y un conjunto de configuraciones por pares ponderadas WPC, definimos la cobertura de $ppCA$, denotada por $cob(ppCA)$, como la suma de los pesos de las configuraciones por pares ponderadas de WPC cubiertas por alguna configuración de $ppCA$ dividida entre la suma de todos los pesos de las configuraciones en WPC, esto es:

$$cob(ppCA) = \frac{\sum_{\substack{wpc \in WPC \\ \exists cc \in ppCA, cc \text{ cubre } wpc.pc}} wpc.w}{\sum_{wpc \in WPC} wpc.w}. \quad (1)$$

El problema de optimización en el que estamos interesados consiste en encontrar una colección de conjuntos de características válidos, $ppCA$, que minimice el número de conjuntos de características $|ppCA|$ y maximice la cobertura $cob(ppCA)$. Este es un problema de optimización bi-objetivo con objetivos contrapuestos. Por tanto, la solución no será única, nuestro objetivo será encontrar un conjunto de soluciones eficientes (no dominadas).

4. Algoritmo Ávido basado en Programación Lineal Entera

Para resolver el problema proponemos usar un algoritmo ávido que, en cada iteración, busca un producto que maximice la cobertura con respecto a la actual. Una vez encontrado, añade dicho producto al conjunto solución, elimina los pares de características cubiertos por el producto y continúa su búsqueda de un nuevo producto. El algoritmo se detiene cuando no es posible aumentar la cobertura, lo cual sucede cuando todos los pares de características con peso mayor que cero han sido cubiertos.

Antes de presentar el algoritmo, describiremos el programa lineal entero que se utiliza como base en cada iteración de APLE.

Sea f el número de características de nuestro modelo fm. Usaremos las variables de decisión $x_j \in \{0, 1\}$ con $j \in \{1, 2, \ldots, f\}$ para indicar si debemos incluir la característica j en el siguiente producto ($x_j = 1$ o no ($x_j = 0$). No todas las combinaciones de características forman productos válidos. Siguiendo a Benavides et al. [9], podemos utilizar una fórmula de lógica proposicional para expresar la validez de un producto en un determinado FM. Para poder incluir

esas restricciones en nuestro programa lineal, esta fórmula se expresará en forma normal conjuntiva (CNF) y se transformarán en desigualdades que se añaden al programa lineal.

A continuación veremos cómo se transforma cada cláusula de la fórmula. Definamos los vectores binarios v y u como sigue:

$$v_j = \begin{cases} 1 \text{ si la característica } j \text{ aparece en la cláusula,} \\ 0 \text{ en otro caso,} \end{cases}$$

$$u_j = \begin{cases} 1 \text{ si la característica } j \text{ aparece negada en la cláusula,} \\ 0 \text{ en otro caso.} \end{cases}$$

Con la ayuda de u y v podemos escribir transformar la cláusula en la siguiente desigualdad para nuestro programa lineal:

$$\sum_{j=1}^{f} v_j(u_j(1-x_j) + (1-u_j)x_j) \geq 1. \tag{2}$$

Por otro lado, necesitaremos variables de decisión para modelar las configuraciones por pares que se cubren con un producto. Las variables las denotaremos con $c_{j,k}$, $c_{j,\overline{k}}$, $c_{\overline{j},k}$ o $c_{\overline{j},\overline{k}}$, dependiendo de la combinación de presencia/ausencia de características en la configuración, y tomarán valor 1 si el producto cubre la configuración y 0 en caso contrario. Los valores de las variables c dependen de los valores de las variables x. Para reflejar esta dependencia en nuestro programa lineal, necesitamos añadir las siguientes restricciones para todos los pares de características $1 \leq j < k \leq f$:

$$2c_{\overline{j},\overline{k}} \leq (1-x_j) + (1-x_k), \tag{3}$$
$$2c_{\overline{j},k} \leq (1-x_j) + x_k, \tag{4}$$
$$2c_{j,\overline{k}} \leq x_j + (1-x_k), \tag{5}$$
$$2c_{j,k} \leq x_j + x_k. \tag{6}$$

En realidad no es necesario añadir todas las variables c posibles, sino solo aquellas que se correspondan con una configuración que no haya sido cubierta anteriormente. Llamemos al conjunto de configuraciones no cubiertas U. Entonces, la función objetivo (a maximizar) de nuestro programa lineal, que es la cobertura conseguida con el producto, tiene como expresión:

$$f(c) = \sum_{(j,k)\in U} w_{j,k}c_{j,k}, \tag{7}$$

donde, abusando de notación, usado j y k para representar características del modelo y su presencia/ausencia, y hemos expresado el peso de las configuración (j,k) con $w_{j,k}$.

En el Algoritmo 1 presentamos nuestra propuesta APLE. En la línea 1 inicializa la lista de productos $ppCA$, que por el momento está vacía. A continuación entra en un bucle en el que busca el producto que maximiza la cobertura con

Algoritmo 1. Algoritmo Ávido basado en Programación Lineal Entera (APLE)

Entrada: U //conjunto de configuraciones con peso mayor que cero
Salida: $ppCA$ // lista de productos
1: $ppCA \leftarrow []$
2: **while** $U \neq \emptyset$ **do**
3: $z \leftarrow$ resolver (mín $f(x)$ sujeto a (2)-(6))
4: $ppCA \leftarrow ppCA + z$
5: $U \leftarrow U/cob(z)$ // Elimina las configuraciones cubiertas por z
6: **end while**

respecto a las configuraciones que quedan por cubrir, U (línea 2). El nuevo producto es añadido a la lista de productos y las configuraciones cubiertas por el producto son eliminadas de U.

5. Evaluación

Esta sección describe como fue llevada a cabo nuestra evaluación. Empezamos describiendo los algoritmos PPGS y pICPL objeto de la comparación, seguido de los métodos usados para asignar prioridades, los modelos de características usados como instancias y la configuración de los experimentos.

5.1. Algoritmo PPGS

El algoritmo llamado *Prioritized Pairwise Genetic Solver* (PPGS) es un algoritmo genético constructivo que sigue un modelo maestro-esclavo para paralelizar la evaluación de los individuos. En cada iteración, el algoritmo añade un nuevo producto al conjunto de pruebas en construcción hasta que todas las combinaciones de características se hayan cubierto. Este algoritmo considera como mejor producto para ser añadido aquel que cubra un conjunto de pares de características (aún por cubrir) que aporten mayor cobertura al conjunto de productos priorizados.

La configuración de parámetros utilizada para PPGS es la siguiente: torneo binario como operador de selección, cruce de un punto con probabilidad $0,8$, mutación con probabilidad $0,1$, población de 10 individuos y condición de parada 1000 evaluaciones de fitness para la generación del siguiente mejor producto. La condición de parada del algoritmo es conseguir cobertura total. Para más detalles consultar [6].

5.2. Algoritmo pICPL

pICPL es un algoritmo ávido que genera matrices de cobertura con fortaleza n (*n-wise covering arrays*) desarrollado por Johansen et al. [3]. Este algoritmo no genera covering arrays con cobertura total sino que cubre sólo aquellas combinaciones que aparecen en al menos un producto priorizado. El resultado obtenido

con este enfoque es equivalente al que se persigue resolviendo este problema, ya que el conjunto de pares por cubrir va a ser el mismo en todos los algoritmos utilizados en este artículo. Debemos destacar que pICPL usa ejecución paralela a nivel de datos. Este paralelismo viene de las operaciones realizadas sobre un conjunto amplio de datos. Para más detalles consultar [3]. Queremos remarcar que existe una versión muy conocida de este algoritmo para testar SPLs, desarrollado por los mismos autores, llamado ICPL [12]. Sin embargo, esa versión no contempla prioridades en la computación del conjunto de pruebas.

5.3. Métodos de Asignación de Prioridades

Consideraremos tres métodos de asignación de pesos a productos:

Valores Medidos Los pesos están derivados de propiedades no funcionales obtenidas de 16 sistemas SPL reales. Estos modelos pertenecen a dominios de problema diferentes, estando implementados usando diferentes tecnologías, y fueron medidos utilizando SPL Conqueror [13]. El resultado son estimaciones reales de propiedades medibles no funcionales como consumo de memoria o rendimiento. Se calculan mediciones para un conjunto de productos, usualmente un subconjunto de aquellos que denota el FM. Esta opción de asignación de pesos nos permite emular escenarios de pruebas más realistas donde los ingenieros de pruebas deben realizar mayor esfuerzo en los productos que demuestran más rendimiento, por ejemplo. Para nuestro trabajo tomamos los valores reales de los productos considerando las interacciones de pares de características. La Tabla 1 resume los sistemas SPL evaluados, su propiedad medida (`Prop`), el número de características o *features* (`NF`), el número de productos (`NP`), número de configuraciones medidas (`NC`), y el porcentaje de productos priorizados (`PP` %) usados en nuestra comparativa, como explicaremos en breve.

Valores Basados en Rango Para esta asignación de pesos seleccionamos los productos a priorizar basados en cómo de diferentes son comparados con otros productos de la SPL y se les asigna un peso basado en su rango de valores. La intuición bajo esta estrategia de asignación es dar un peso similar a dos productos muy diferentes. De esta manera los pesos altos están esparcidos entre un número alto de pares de características haciendo que el conjunto de pruebas priorizado sea más difícil de computar. Además, esto nos da la posibilidad de seleccionar diferentes porcentajes de productos usados en el cálculo final de los pesos de las características, como explicaremos a continuación.

Valores Aleatorios Esta asignación de pesos se consigue generando valores aleatorios dentro del rango obtenido en el enfoque anteriormente explicado (Valores Basados en Rango).

Vamos a seleccionar los productos para priorizar basándonos en los métodos de asignación. Para el método de *Valores Medidos*, todos los productos medidos

SPL Name	**Prop**	**NF**	**NP**	**NC**	**PP %**
Prevayler	F	6	32	24	75.0
LinkedList	F	26	1440	204	14.1
ZipMe	F	8	64	64	100.0
PKJab	F	12	72	72	100.0
SensorNetwork	F	27	16704	3240	19.4
BerkeleyDBF	F	9	256	256	100.0
Violet	F	101	≈ 1E20	101	≈ 0.0
Linux subset	F	25	≈ 3E24	100	≈ 0.0
LLVM	M	12	1024	53	5.1
Curl	M	14	1024	68	6.6
x264	M	17	2048	77	3.7
Wget	M	17	8192	94	1.15
BerkeleyDBM	M	19	3840	1280	33.3
SQLite	M	40	≈ 5E7	418	≈ 0.0
BerkeleyDBP	P	27	1440	180	12.50
Apache	P	10	256	192	75.0

Tabla 1. Resumen de los Valores Medidos para los Casos de Estudio

fueron usados como productos priorizados. Para los métodos *Basados en Rango* y *Aleatorio*, sólo un porcentaje de los productos denotados por cada FM fue usado como producto priorizado. Los porcentajes usados fueron: 5 %, 10 %, 20 %, 30 % y 50 %.

5.4. Casos de Estudio

Vamos a usar tres grupos de instancias basados en el número de productos denotados por el FM y como fueron asignadas las prioridades, como se muestra en la Tabla 2. El grupo G1 está compuesto por 160 FMs, cuyo número de productos oscila entre 16 y 1000 productos, y fueron evaluados usando los métodos *Basados en Rango* y *Aleatorio*. El grupo G2 está compuesto por 59 FMs con rango de productos entre 1000 y 80000 productos, y fueron evaluados usando estos mismos dos métodos de asignación. El grupo G3 está formado por 16 FMs reales, con un número de productos entre 16 y ≈3E24, que fueron evaluados usando el método de valores medidos. En este estudio analizamos un total de 235 FMs que fueron extraídos de varias fuentes: SPL Conqueror, Johansen et al. [3], y el repositorio SPLOT [8]. Para G1 y G2 las instancias son calculadas usando para cada FM dos métodos de asignación de prioridades a los productos y tres porcentajes diferentes de selección de productos priorizados. Mientras que en G3, sólo usamos el método de asignación de valores medidos. Esto hace un total de 1330 instancias analizadas para cada uno de los tres algoritmos de la comparación.

	G1	G2	G3	Resumen
NFM	160	59	16	235
NP	16-1K	1K-80K	32- ≈3E24	16- ≈3E24
NC	10-56	14-67	6-101	6-101
MA	R,A	R, A	M	
PP %	20,30,50	5,10,20	≈0.0 – 100	
IP	960	354	16	1330

NFM: Número de FMs, **NP**: Numero de Productos, **NC**: Numero de Características, **MA**: Método Asignación, **R**: Rango, **A**: Aleatorio, **M**: Medido, **PP %**: Porcentaje de Productos Priorizados, **IP**: Instancias del Problema

Tabla 2. Resumen de los Casos de Estudio

5.5. Configuración de los Experimentos

PPGS y pICPL son algoritmos no deterministas, por tanto hemos ejecutado 30 veces estos algoritmos para realizar una comparación justa entre ellos. Por otra parte, nuestro enfoque APLE es un algoritmo determinista y fue ejecutado una sola vez por cada instancia. Como medida de rendimiento de los algoritmos hemos analizado el número de productos requerido para alcanzar un porcentaje de cobertura ponderada dada y el tiempo de computación requerido. En ambos casos queremos minimizar los objetivos, ya que queremos conjuntos de prueba más pequeños obtenidos en el menor tiempo posible.

Las ejecuciones de PPGS y pICPL se ejecutaron en un clúster con máquinas Intel Core2 Quad processors Q9400 de 4 núcleos por procesador, por tanto su nivel de paralelismo es de 4 hebras para cada ejecución independiente. APLE fue ejecutado en un sólo núcleo ya que no es un algoritmo paralelo, esto debe ser teniendo en cuenta en la interpretación de los tiempos de ejecución obtenidos.

6. Análisis de los Resultados

En este trabajo usamos dos técnicas estadísticas para medir diferentes aspectos de la comparativa.

6.1. Test de Kruskal-Wallis

Para comprobar si las diferencias entre los algoritmos son estadísticamente significativas hemos aplicado el test no paramétrico Kruskal-Wallis usando la corrección de Bonferroni. En las tablas destacamos con un gris oscuro que un valor es estadísticamente significativo con respecto a los otros dos algoritmos y con gris claro cuando existen diferencias con sólo uno. El nivel de confianza utilizado es el habitual 95 % (p-value menor que $0,05$).

En la Tabla 3 resumimos los resultados obtenidos para el grupo 1 de FMs con hasta 1000 productos. Cada columna corresponde a un algoritmo y en las filas mostramos el número de productos necesarios para alcanzar desde 50 % a 100 %

de cobertura ponderada. Los datos de cada celda son la media y la desviación típica de todas las ejecuciones de las instancias de G1. Podemos observar que los resultados de APLE son estadísticamente diferentes para todos los valores de cobertura y tiempo de ejecución con respecto a al menos uno de los otros algoritmos. Concretamente en 7 ocasiones es estadísticamente diferente con respecto a los otros dos algoritmos. Aparentemente los resultados son estadísticamente mejores cuando las diferencias existen, hecho que confirmaremos en la siguiente subsección aplicando otro test estadístico. Queremos destacar también la gran ventaja con respecto al tiempo de computación de PPGS.

Cobertura	APLE	PPGS	pICPL	Cobertura	APLE	PPGS	pICPL
50 %	$1,19_{0,39}$	$1,20_{0,40}$	$1,20_{0,40}$	96 %	$3,98_{1,21}$	$4,00_{1,23}$	$4,37_{1,42}$
75 %	$1,91_{0,51}$	$1,92_{0,51}$	$1,98_{0,58}$	97 %	$4,35_{1,31}$	$4,38_{1,32}$	$4,71_{1,54}$
80 %	$2,13_{0,58}$	$2,15_{0,59}$	$2,25_{0,68}$	98 %	$4,80_{1,42}$	$4,83_{1,46}$	$5,18_{1,74}$
85 %	$2,44_{0,71}$	$2,47_{0,72}$	$2,58_{0,81}$	99 %	$5,53_{1,64}$	$5,58_{1,71}$	$5,87_{1,99}$
90 %	$2,86_{0,85}$	$2,88_{0,86}$	$3,13_{1,03}$	100 %	$7,49_{2,79}$	$7,56_{2,85}$	$7,56_{3,03}$
95 %	$3,70_{1,13}$	$3,72_{1,14}$	$4,06_{1,33}$	Tiempo	1562_{2026}	23897_{28669}	10116_{18842}

Tabla 3. Media y desviación estándar para las instancias del grupo G1. El tiempo está medido en milisegundos.

La Tabla 4 muestra los resultados para el grupo G2 de FMs de entre 1000 y 80000 productos. En este caso las diferencias significativas siempre existen entre APLE y PPGS con respecto a pICPL. En otras palabras, ambos algoritmos son mejores que pICPL en la calidad del conjunto de pruebas generado. Nuevamente APLE es significativamente mejor que los otros dos algoritmos en 7 ocasiones.

Con respecto al tiempo de computación, APLE es más rápido que los otros algoritmos con una amplia diferencia. En este caso su media de tiempo empleado es de 5 segundos, frente a los alrededor de 6 minutos para pICPL y los más de 10 minutos empleados por PPGS.

Cobertura	APLE	PPGS	pICPL	Cobertura	APLE	PPGS	pICPL
50 %	$1,16_{0,37}$	$1,16_{0,36}$	$1,36_{0,83}$	96 %	$4,96_{0,97}$	$4,98_{0,97}$	$5,83_{3,14}$
75 %	$2,08_{0,41}$	$2,09_{0,42}$	$2,47_{1,65}$	97 %	$5,50_{1,08}$	$5,55_{1,10}$	$6,43_{3,27}$
80 %	$2,36_{0,54}$	$2,39_{0,52}$	$2,86_{1,79}$	98 %	$6,27_{1,26}$	$6,34_{1,34}$	$7,23_{3,48}$
85 %	$2,73_{0,62}$	$2,73_{0,59}$	$3,27_{2,08}$	99 %	$7,48_{1,74}$	$7,66_{1,88}$	$8,59_{4,11}$
90 %	$3,32_{0,80}$	$3,36_{0,76}$	$3,98_{2,38}$	100 %	$13,89_{9,85}$	$14,57_{10,65}$	$13,79_{9,98}$
95 %	$4,55_{0,92}$	$4,59_{0,90}$	$5,42_{3,12}$	Tiempo	$4900_{6,7E+3}$	$273728_{7,2E+5}$	$638164_{2,1E+6}$

Tabla 4. Media y desviación estándar para las instancias del grupo G2. El tiempo está medido en milisegundos.

En la Tabla 5 se muestran resultados del grupo de instancias G3, donde las diferencias significativas que existen son algo menores, pero las conclusiones son similares a las obtenidas con los anteriores grupos de instancias, en la mayoría de los casos existen diferencias entre APLE y pICPL. Con respecto a PPGS, los resultados en calidad de la solución esta vez son similares, pero el tiempo de computación sigue siendo mucho menor para APLE.

Cobertura	APLE	PPGS	pICPL	Cobertura	APLE	PPGS	pICPL
50 %	$1,56_{0,51}$	$1,58_{0,49}$	$1,56_{0,50}$	96 %	$5,69_{1,19}$	$5,86_{1,18}$	$6,38_{1,58}$
75 %	$2,63_{0,81}$	$2,66_{0,77}$	$2,75_{0,75}$	97 %	$6,13_{1,26}$	$6,24_{1,38}$	$6,75_{1,39}$
80 %	$2,81_{0,83}$	$2,81_{0,73}$	$3,25_{0,97}$	98 %	$6,81_{1,47}$	$6,98_{1,55}$	$7,44_{1,66}$
85 %	$3,44_{0,89}$	$3,46_{0,87}$	$3,81_{0,95}$	99 %	$7,75_{1,69}$	$7,92_{1,87}$	$8,75_{2,08}$
90 %	$4,06_{1,06}$	$4,12_{1,04}$	$4,56_{1,27}$	100 %	$11,69_{5,69}$	$12,08_{6,50}$	$12,19_{5,68}$
95 %	$5,38_{1,09}$	$5,45_{1,14}$	$6,06_{1,44}$	Tiempo	$18262_{1,0E+4}$	$2048719_{7,6E+6}$	$24330_{5,9E+4}$

Tabla 5. Media y desviación estándar para las instancias del grupo G3. El tiempo está medido en milisegundos.

Como conclusión general de este primer análisis podemos decir que APLE nunca es estadísticamente peor que ninguno de los otros algoritmos de la comparativa, siendo claramente mejor en muchos de los casos estudiados y en tiempo de computación. Para los grupos de instancias estudiados siempre es aconsejable el uso de APLE para la computación de conjuntos de pruebas priorizadas.

6.2. Estadístico $\hat{A}_{12}$

La interpretación de los test estadísticos puede ser engañosa en alguno de los casos, por eso es siempre aconsejable reportar una medida del tamaño del efecto. Para este propósito hemos usado el estadístico $\hat{A}_{12}$ como recomiendan Arcuri y Briand [14]. Dada una medida de rendimiento M, $\hat{A}_{12}$ mide la probabilidad de que un algoritmo A consiga valores más altos para M que otro algoritmo B. Si los dos algoritmos son equivalentes, entonces $\hat{A}_{12} = 0,5$. Si $\hat{A}_{12} = 0,3$ entonces vamos a obtener valores más pequeños en el 70 % de las ocasiones con el algoritmo A.

A continuación mostramos los valores del estadístico $\hat{A}_{12}$ para medir la significancia práctica de los resultados. Las tablas comparan los resultados de los algoritmos dos a dos para los tres grupos de instancias. En estas tablas un valor menor a 0,5 indica mejores resultados para el primer algoritmo de la comparación, siendo las tablas las comparaciones de APLE-pICPL y APLE-PPGS.

En la Tabla 6 podemos observar como en ninguna ocasión pICPL tiene una probabilidad mayor que APLE de obtener un mejor conjunto de pruebas. En el peor caso, para el grupo G3 y 50 % de cobertura tenemos un empate en probabilidad. En 32 combinaciones de grupo y cobertura es más probable que APLE consiga una mejor solución, siendo el porcentaje más favorable un 79,7 %. Este estadístico no deja duda de que las diferencias significativas son en favor de APLE.

Grupo	50 %	75 %	80 %	85 %	90 %	95 %	96 %	97 %	98 %	99 %	100 %
G1	0,4953	0,4693	0,4380	0,4370	0,3672	0,3417	0,3286	0,3526	0,3526	0,3865	0,4870
G2	0,4492	0,4096	0,3588	0,3573	0,3121	0,2641	0,2669	0,2387	0,2288	0,2133	0,3446
G3	0,5000	0,4375	0,3125	0,3438	0,3125	0,2812	0,3125	0,2812	0,3438	0,2812	0,3750

Tabla 6. Resultado del estadístico $\hat{A}_{12}$ entre APLE-pICPL para todos los grupos

La Tabla 7 muestra nuevamente que la probabilidad de obtener conjuntos de pruebas mínimos es mayor para APLE que para PPGS. Tan sólo en dos ocasiones la probabilidad es mayor para PPGS, son las celdas G2-50 % y G3-80 %, aunque las diferencias son muy escasas, 50,26 % y 50,42 % respectivamente. En el resto de las 31 combinaciones de grupo y cobertura las probabilidades son favorables a APLE. Sin duda, esta prueba estadística también confirma que las diferencias significativas existentes entre APLE y PPGS son favorables a APLE.

Grupo	50 %	75 %	80 %	85 %	90 %	95 %	96 %	97 %	98 %	99 %	100 %
G1	0,4970	0,4967	0,4881	0,4912	0,4921	0,4936	0,4924	0,4889	0,4918	0,4910	0,4810
G2	0,5026	0,4943	0,4868	0,4936	0,4750	0,4745	0,4832	0,4738	0,4653	0,4288	0,3693
G3	0,4938	0,4854	0,5042	0,4917	0,4750	0,4708	0,4188	0,4479	0,4354	0,4417	0,4146

Tabla 7. Resultado del estadístico $\hat{A}_{12}$ entre APLE-PPGS para todos los grupos

7. Amenazas a la Validez

Hemos identificado dos amenazas a la validez de este trabajo. La primera es que la selección de los modelos de características que forman parte de nuestro experimento puede afectar a los resultados obtenidos. Para combatir esta amenaza hemos usado tres fuentes diferentes para obtener un gran número de FMs, con un amplio rango tanto de número de características como de número de productos que denotan. La segunda es la selección de los métodos de asignación de prioridades. Para aliviar este problema hemos utilizado tres métodos diferentes de asignación de prioridades y diferentes porcentajes de productos tenidos en cuenta. Además, para el grupo G3 estas prioridades están calculadas sobre propiedades no funcionales del propio código fuente, por lo que la distribución de pesos utilizada para ponderar las características es realista.

8. Conclusiones y Trabajo Futuro

En este artículo hemos presentado un enfoque novedoso que hibrida un algoritmo ávido con un resolutor ILP. Nuestra propuesta algorítmica se ha evaluado con 235 FMs con diferentes características y usando diferentes criterios de selección para la ponderación de los productos. Hemos comparado APLE con PPGS y pICPL, siendo APLE siempre superior manifiestamente en rendimiento, tanto en tiempo de ejecución como en calidad de la solución, especialmente frente a pICPL. Por tanto, nuestra recomendación es usar APLE para el cálculo de conjunto de pruebas priorizadas para SPLs, ya que consigue mantener la tasa de detección de fallos usando menos productos de prueba.

Nuestra propuesta, APLE, no asegura que vaya a obtener el frente de Pareto del problema bi-objetivo que pretendemos resolver, ya que en cada iteración solo decide cuál será el siguiente producto a añadir. Para obtener el frente de Pareto sería necesario buscar un conjunto de productos en cada iteración. Algunos resultados preliminares que hemos obtenido en esta línea sugieren que el

número de variables y restricciones de las instancias ILP necesarias para resolver el problema crecen muy rápido con el número de productos, y esto repercute negativamente en el tiempo de requerido por el resolutor ILP para resolver el problema. Sería interesante analizar este incremento del número de variables para buscar alternativas para obtener el frente de Pareto en un tiempo razonable.

Acknowledgements

Esta investigación ha sido parcialmente financiada por el Ministerio de Economía y Competitividad y fondos FEDER (proyecto TIN2014-57341-R), la beca BES-2012-055967, la Universidad de Málaga y Andalucía Tech.

Referencias

1. Pohl, K., Bockle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer (2005)
2. Cichos, H., Oster, S., Lochau, M., Schürr, A.: Model-based coverage-driven test suite generation for software product lines. In: MoDELS. (2011) 425–439
3. Johansen, M.F., Haugen, Ø., Fleurey, F., Eldegard, A.G., Syversen, T.: Generating better partial covering arrays by modeling weights on sub-product lines. In France, R.B., Kazmeier, J., Breu, R., Atkinson, C., eds.: MoDELS. Volume 7590 of Lecture Notes in Computer Science., Springer (2012) 269–284
4. Engström, E., Runeson, P.: Software product line testing - a systematic mapping study. Information & Software Technology **53**(1) (2011) 2–13
5. da Mota Silveira Neto, P.A., do Carmo Machado, I., McGregor, J.D., de Almeida, E.S., de Lemos Meira, S.R.: A systematic mapping study of software product lines testing. Information & Software Technology **53**(5) (2011) 407–423
6. Lopez-Herrejon, R.E., Javier Ferrer, J., Chicano, F., Haslinger, E.N., Egyed, A., Alba, E.: A parallel evolutionary algorithm for prioritized pairwise testing of software product lines. GECCO '14, New York, NY, USA, ACM (2014) 1255–1262
7. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University (1990)
8. Mendonca, M.: Software Product Line Online Tools(SPLOT) (2013) http://www.splot-research.org/.
9. Benavides, D., Segura, S., Cortés, A.R.: Automated analysis of feature models 20 years later: A literature review. Inf. Syst. **35**(6) (2010) 615–636
10. Cohen, M.B., Dwyer, M.B., Shi, J.: Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. IEEE Trans. Software Eng. **34**(5) (2008) 633–650
11. Stevens, B., Mendelsohn, E.: Efficient software testing protocols. In: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research. CASCON '98, IBM Press (1998) 22–
12. Johansen, M.F., Haugen, Ø., Fleurey, F.: An algorithm for generating t-wise covering arrays from large feature models. In: SPLC (1). (2012) 46–55
13. Siegmund, N., Rosenmüller, M., Kästner, C., Giarrusso, P., Apel, S., Kolesnikov, S.: Scalable prediction of non-functional properties in software product lines: Footprint and memory consumption. Information & Soft. Technology **55**(3) (2013) 491–507
14. Arcuri, A., Briand, L.: A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Softw. Test. Verif. Reliab (2012)

Estudio de mecanismos de hibridación para el descubrimiento evolutivo de arquitecturas

Aurora Ramírez, José Antonio Molina, José Raúl Romero y Sebastián Ventura

Dpto. de Informática y Análisis Numérico
Universidad de Córdoba, Campus de Rabanales, 14071 Córdoba, España
{aramirez, i12momej, jrromero, sventura}@uco.es

Resumen Las decisiones que los ingenieros software toman durante el análisis arquitectónico pueden verse influenciadas por aspectos como la naturaleza del sistema bajo estudio o los criterios de calidad que deben guiar su desarrollo, generalmente expresados en términos de métricas software. A la hora de abordar tareas de diseño arquitectónico como problemas de optimización, factores como los anteriores también deben ser tenidos en cuenta, ya que podrían afectar al funcionamiento de cualquier algoritmo de búsqueda. Incluir técnicas de búsqueda local en un algoritmo evolutivo constituye un mecanismo habitual para intentar mejorar su rendimiento, si bien diseñar un modelo híbrido añade nuevas variables a ser estudiadas. En este trabajo se analiza la idoneidad de este tipo de enfoque para la resolución del problema del descubrimiento de arquitecturas software. El estudio experimental realizado muestra que las características del problema pueden influir tanto en la eficiencia de la búsqueda local como en la calidad de las soluciones obtenidas.

Keywords: ingeniería del software basada en búsqueda, arquitecturas software, algoritmos evolutivos, búsqueda local

1. Introducción

Durante la fase de análisis de software, los arquitectos software deben tomar complejas decisiones de diseño con el fin de obtener un sistema que no sólo cumpla los requisitos funcionales, sino que también satisfaga múltiples criterios de calidad. En situaciones donde las habilidades y experiencias del ingeniero resultan determinantes, disponer de sistemas de apoyo a la decisión puede aportar grandes beneficios, ya que ofrecen al ingeniero más alternativas de diseño.

En este sentido, la Ingeniería del Software basada en búsqueda (*Search Based Software Engineering*, SBSE) [1] puede verse como una forma de asistir al ingeniero en su desempeño diario, ya que propone la aplicación de técnicas eficientes de búsqueda y optimización para resolver tareas propias de la Ingeniería del Software. Técnicas metaheurísticas [2], como son los algoritmos evolutivos, han sido empleadas con éxito en la resolución de problemas tan diversos como la priorización de requisitos o la selección de casos de prueba. Sin embargo, la elección de

un tipo de algoritmo u otro no siempre es justificada en base a las características del problema, quedando dicha decisión fundamentada en aspectos como la popularidad de ciertos métodos o la disponibilidad de sus implementaciones.

Dentro del diseño automático de software [3] se engloban los problemas de optimización arquitectónica [4], los cuales plantean la necesidad de identificar o mejorar diseños arquitectónicos a partir de artefactos abstractos tales como componentes, servicios o patrones. El descubrimiento de arquitecturas software es un ejemplo de este tipo de problemas, donde el objetivo es extraer la arquitectura basada en componentes de un sistema a partir de modelos de análisis más detallados, como son los diagramas de clases [5]. Hasta el momento, su resolución ha sido abordada mediante algoritmos evolutivos guiados por la medición de métricas software relacionadas con criterios de calidad como la modularidad y la reutilización. A pesar de obtener resultados satisfactorios con la mayoría del software analizado, cabe preguntarse si la eficiencia de la búsqueda puede verse mejorada, y en qué aspectos, a la hora de abordar sistemas más complejos.

Un mecanismo ampliamente reconocido por su capacidad para mejorar el rendimiento de un algoritmo de búsqueda es la hibridación con otras técnicas de optimización [6]. Este tipo de modelos se fundamentan en la idea de que explotar las características propias de diferentes métodos va a permitir obtener resultados más satisfactorios allí donde las técnicas *puras* encuentran dificultades. Uno de los mecanismos de hibridación más empleados para mejorar la eficiencia de los algoritmos evolutivos se basa en combinarlos con técnicas de búsqueda local, dando lugar a los llamados *algoritmos meméticos*. Gracias a la introducción de la búsqueda local, un método de búsqueda global como es un algoritmo evolutivo va a poder explotar mejor el entorno de vecindad de las soluciones que conforman la población, llegando por lo general a mejores soluciones y en un tiempo menor [7].

No obstante, determinar cuál es el modo más efectivo de integrar ambos métodos de optimización no resulta sencillo, ya que se debe establecer un balance adecuado en cuanto a los recursos destinados a cada uno de ellos. En este sentido, un exceso de iteraciones en la búsqueda local podría implicar que el algoritmo evolutivo no pueda explorar otras zonas del espacio de búsqueda. Por otro lado, es también preciso considerar aspectos como cuándo aplicar la búsqueda local, sobre qué soluciones hacerlo o qué algoritmo concreto emplear. Todo ello hace que el diseño de un algoritmo memético resulte muy complejo, pues estos factores también pueden verse afectados por el problema de optimización bajo estudio [7].

Ante la imposibilidad de conocer a priori el rendimiento de un modelo híbrido sobre el problema del descubrimiento de arquitecturas software, es conveniente plantear el diseño de diversas variantes y comparar su rendimiento. Es por ello que este trabajo presenta un estudio comparativo de dos modelos meméticos, uno donde la búsqueda local actúa como un operador genético y otro donde se utiliza como mecanismo de refinamiento sobre el resultado devuelto por el algoritmo evolutivo. A su vez, para cada uno de ellos se analiza la influencia de los parámetros que lo definen, tanto desde el punto de vista evolutivo como de aspectos del dominio del problema, como son las medidas software a optimizar.

El resto del artículo se estructura como sigue. La Sección 2 recopila el trabajo relacionado, centrándose en los conceptos que rodean al desarrollo de algoritmos meméticos y su uso en SBSE. A continuación, la Sección 3 detalla el diseño de los modelos híbridos a comparar a partir del algoritmo evolutivo inicialmente propuesto. El estudio experimental se lleva a cabo en la Sección 4, donde se analizan los resultados y se discute el rol de la búsqueda local desde diferentes perspectivas. Finalmente, la Sección 5 recoge las conclusiones y el trabajo futuro.

2. Trabajo relacionado

Los modelos híbridos surgen de la necesidad de mejorar la eficiencia de los métodos de resolución *puros* en aquellos contextos donde éstos encuentran dificultades gracias a que pueden verse reforzados allí donde otras técnicas de optimización están especializadas [6]. Posiblemente, la idea de integrar un método de búsqueda local como la escalada o el enfriamiento simulado [2] en un algoritmo evolutivo sea uno de los mecanismos de hibridación más conocidos y estudiados, pues es sabido que los algoritmos poblacionales tienen una mejor capacidad para explorar el espacio de búsqueda que para explotarlo.

Por el contrario, las técnicas de búsqueda local promueven la intensificación por encima de la diversidad [8], ya que parten de una única solución para la que exploran su entorno de vecindad. El modelo más simple es la búsqueda en escalada (*Hill Climbing*, HC), un método iterativo que sólo permite realizar movimientos a soluciones mejores que la actual. Puesto que este método puede verse atrapado en óptimos locales, técnicas como el enfriamiento simulado (*Simulated Annealing*, SA) permiten aceptar movimientos hacia peores soluciones al comienzo del proceso e ir aumentando progresivamente la intensificación.

Respecto a la integración de ambos paradigmas, suele entenderse que un algoritmo memético implica realizar la búsqueda local tras la ejecución de los operadores genéticos. No obstante, existen definiciones menos restrictivas donde el punto concreto de hibridación es considerado como una decisión de diseño más [7]. Desde esta perspectiva, los aspectos que deben definirse a la hora de diseñar un algoritmo memético son los siguientes:

- El balance entre la búsqueda local y global, entendida como el número de iteraciones o evaluaciones destinadas a cada una de ellas.
- El algoritmo de búsqueda local a utilizar.
- El momento y la frecuencia con la que se aplica la búsqueda local.
- El conjunto de individuos sobre los que se realiza la búsqueda local y el mecanismo para seleccionarlo.

A pesar de que los algoritmos meméticos gozan de cierta popularidad como métodos eficientes de búsqueda, no es frecuente encontrar este tipo de propuestas en el ámbito de SBSE. Posiblemente esto sea debido a la dificultad que conlleva su diseño, ya que requieren un conocimiento profundo de las técnicas existentes. Centrándose en problemas de diseño software, en [9] se presenta una comparación entre algoritmos evolutivos y de colonias de hormigas para el diseño de clases.

Los autores destacan que la incorporación de la búsqueda local, representada aquí mediante un algoritmo voraz, no siempre resulta beneficiosa aunque permita generar soluciones suficientemente buenas en un tiempo menor. Por otro lado, el trabajo realizado en [10] analiza la influencia de SA a la hora de especificar arquitecturas software en términos de clases y patrones de diseño. Los resultados presentados muestran que el orden en el que se combina la búsqueda local con el algoritmo evolutivo tiene un efecto notable en los resultados.

3. Modelos meméticos propuestos

En esta sección se presentan el problema de optimización a resolver y el algoritmo evolutivo propuesto inicialmente. Dicho algoritmo constituye el punto de partida para el diseño de los dos modelos meméticos explicados a continuación.

3.1. Descripción del problema de búsqueda

El descubrimiento de arquitecturas software basadas en componentes puede tratarse como un problema de búsqueda combinatorio en el cual se pretende extraer la estructura en componentes de un sistema a partir de su diagrama de clases [5]. Cada componente se construye a partir de una agrupación óptima de clases, esto es, clases que están muy relacionadas entre sí. Las relaciones existentes entre clases alojadas en distintos componentes permitirán construir las interfaces, mientras que los conectores se identifican a partir de parejas de interfaces requeridas y proveídas.

Los criterios de evaluación sobre el problema hacen referencia a aspectos de cohesión y acoplamiento, así como al tamaño de los componentes generados. A su vez, el problema presenta una serie de restricciones. Las arquitecturas resultantes no pueden tener clases duplicadas ni componentes vacíos. Tampoco se permite la existencia de componentes aislados (no definen ninguna interfaz) o mutuamente dependientes.

3.2. Algoritmo evolutivo inicial

El algoritmo evolutivo propuesto en [5] parte de una población de soluciones generadas aleatoriamente, de forma que las clases del modelo de entrada se distribuyen de manera aleatoria en un número también aleatorio de componentes. Cada solución es codificada mediante una estructura en árbol, similar a la que se utiliza en las herramientas de modelado.

La evaluación de los individuos se realiza a partir de varias medidas software, cuya formulación se muestra en la Tabla 1. ICD, que debe ser maximizada, calcula la media del ratio entre relaciones internas (CI^{in}) y externas (CI^{out}) de cada componente, i, considerando también el número de clases que lo componen (cl_i) y el número total de clases (cl_t). A continuación, ERP establece una penalización al contabilizar el número de relaciones entre clases pertenecientes a distintos componentes, i y j, que no pueden ser especificadas como interfaces. Esta

Tabla 1: Medidas software para evaluar las soluciones arquitectónicas

Medida	**Formulación**
ICD: Intra-modular Coupling Density	$ICD_i = ((\#cl_t - \#cl_i)/\#cl_t) \cdot \left(CI_i^{in}/(CI_i^{in} + CI_i^{out})\right)$ $ICD = \sum_{i=1}^{n} ICD_i/n$
ERP: External Relations Penalty	$ERP = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (w_{as} \cdot n_{as_{ij}} + w_{ag} \cdot n_{ag_{ij}}$ $+w_{co} \cdot n_{co_{ij}} + w_{ge} \cdot n_{ge_{ij}})$
CS: Critical Size	$CC_i = 1$ *si* $\#cl_i > umbral$, 0 *en otro caso* $CS = \sum_{i=1}^{n} CC_i$
CB: Component Balance	$SB(n) = \frac{n-\gamma}{\mu-\gamma}$ *si* $n < \mu$, $= 1 - \frac{n-\mu}{\omega-\mu}$ *si* $\mu < n < \omega$, $= 0$ *si* $n \geq \omega$ $CSU(n) = 1 - Gini(\{\#cl_i \forall i \in [1, n]\})$, $CB = SB(n) \cdot CSU$

medida, que debe minimizarse, diferencia entre asociaciones (n_{as}), agregaciones (n_{ag}), composiciones (n_{co}) y generalizaciones (n_{ge}), donde cada una de ellas tiene asociado un peso (w). La medida CS considera el número de componentes que exceden un tamaño crítico ($umbral$), por lo que también debe minimizarse. Finalmente, CB establece un balance entre el número de componentes (n) y su tamaño, debiendo ser maximizada. Los coeficientes γ, ω, y μ representan el número mínimo, máximo y medio de componentes, respectivamente.

A partir de las cuatro medidas anteriores se obtiene el *fitness* de cada solución, s, como la suma de la posición que ocupa el individuo al ordenar toda la población para cada una de las medidas (función r), tal y como se muestra en la Ecuación 1. Si la solución no cumple las restricciones establecidas en la Sección 3.1, se le asigna un ranking superior al de cualquier solución válida.

$$f(s) = \begin{cases} r(ICD(s)) + r(ERP(s)) + r(CS(s)) + r(CB(s)) & \text{si } s \text{ es } válida \\ tamPobl \cdot \#medidas + 1 & \text{si } s \text{ es } inválida \end{cases} \quad (1)$$

Finalmente, el proceso iterativo de búsqueda, que concluye cuando se alcanza un número máximo de evaluaciones, consta de las siguientes fases:

- Selección de padres: Se utiliza un torneo binario sobre la población actual para elegir las soluciones a las que se le aplicará el operador de mutación.
- Mutación de individuos: Los individuos previamente seleccionados son transformados mediante un operador de mutación que puede realizar hasta cinco acciones diferentes: crear, eliminar, mezclar o dividir componentes, así como mover clases de un componente a otro. La selección del mecanismo a aplicar se realiza en base a una ruleta probabilística ponderada.
- Reemplazo de la población: La población de la siguiente generación se construye manteniendo al 10 % de los mejores individuos de la generación actual y añadiéndole los mejores descendientes hasta alcanzar el tamaño poblacional.

3.3. Algoritmo con búsqueda local como operador

El primer modelo memético, denominado EA(LS), incluye un procedimiento de búsqueda local dentro del ciclo iterativo del algoritmo descrito en la Sección 3.2, esto es, considera la búsqueda local como si de un operador genético se tratase. La búsqueda local tiene como objetivo explorar el entorno de vecindad de una solución dada. Para el problema aquí tratado, dicho entorno se ha definido como el formado por aquellas soluciones arquitectónicas que difieren de la solución de partida en la ubicación de una única clase. Por tanto, las soluciones vecinas se pueden obtener mediante la aplicación del operador de mutación *mover clase*. Por otro lado, en cada iteración de la búsqueda local se va a generar una única solución con la que comparar la solución actual. Ambas decisiones vienen condicionadas por la propia naturaleza del problema, ya que el resto de mutaciones generan soluciones muy diferentes a la inicial (lo cual no encaja con el concepto de vecindad), mientras que generar y evaluar todo el entorno de vecindad de una solución consumiría un número excesivo de evaluaciones.

En lugar de fijar el número de iteraciones que van a ejecutarse durante la búsqueda local, se pretende aquí establecer dicho valor dinámicamente conforme avanza la búsqueda. Este enfoque permite un mejor control en cuanto al consumo de evaluaciones, obteniendo por lo general algoritmos más efectivos [11]. En concreto, se ha considerado que el número de iteraciones venga determinado por el número de evaluaciones ya consumidas, de forma que se permiten más iteraciones conforme avanza la búsqueda, pues se espera que las soluciones generadas entonces sean más interesantes. La Ecuación 2 define la función logística con la que se obtiene el número de iteraciones, cuyo valor estará siempre comprendido entre un mínimo ($minIterLS$) y un máximo ($maxIterLS$) configurables. El parámetro k permite adaptar la forma de la curva de la función resultante, la cual devuelve el punto medio del rango establecido entre $minIterLS$ y $maxIterLS$ cuando el algoritmo ha consumido la mitad de las evaluaciones.

$$nIterLS = \frac{maxIterLS - minIterLS}{1 + e^{-k \cdot (nEval - maxEval/2)}} \tag{2}$$

El Algoritmo 1 detalla el algoritmo memético resultante, el cual sigue los mismos pasos y proceso de evaluación que el algoritmo evolutivo original hasta el momento en el que se realiza la mutación (línea 7). A continuación, se calculan las medidas software de la Tabla 1 sobre las nuevas soluciones (línea 8), y se incrementa el contador de evaluaciones convenientemente (línea 9). La generación de nuevos individuos implica que deben reasignarse los rankings (línea 10), si bien se ha considerado que este proceso no consume evaluaciones. Cada uno de los individuos mutados es sometido a una búsqueda local (líneas 11-17) con una cierta probabilidad ($probLS$) y un número de iteraciones determinado por la Ecuación 2 (línea 13). Nótese que dicho número coincide con el número de evaluaciones consumidas por la búsqueda local para ese individuo. Por otro lado, la decisión de si la solución vecina debe reemplazar a la actual se basa en comprobar si la nueva solución obtendría un ranking menor (mejor *fitness*)

con respecto a la población actual (*poblacion*). Finalmente, el algoritmo vuelve a calcular los rankings (línea 18) con el fin de realizar el reemplazo (línea 19).

Algoritmo 1 Pseudocódigo del modelo memético EA(LS)

Require: $maxEval$, $tamPobl$, $probLS$, $minIterLS$, $maxIterLS$
Ensure: $mejorIndividuo$
1: $poblacion \leftarrow$ inicializarPoblacion($tamPobl$)
2: evaluarMedidas($poblacion$)
3: $nEval \leftarrow tamPobl$
4: asignarRankings($poblacion$)
5: **while** $nEval \leq maxEval$ **do**
6: $padres \leftarrow$ seleccionarPadres($poblacion$)
7: $descencientes \leftarrow$ mutarIndividuos($padres$)
8: evaluarMedidas($descendientes$)
9: $nEval \leftarrow nEval + tamPobl$
10: asignarRankings($poblacion \cup descendientes$)
11: **for all** $individuo \in descendientes$ **do**
12: **if** aleatorio(0,1) $\geq probLS$ **then**
13: $nIterLS \leftarrow$ calcularNumIter($minIterLS$, $maxIterLS$, $nEval$, $maxEval$)
14: $individuo \leftarrow$ busquedaLocal($individuo$, $nIterLS$, $poblacion$)
15: $nEval \leftarrow nEval + nIterLS$
16: **end if**
17: **end for**
18: asignarRankings($poblacion \cup descendientes$)
19: $poblacion \leftarrow$ reemplazo($poblacion \cup descendientes$)
20: **end while**
21: **return** seleccionarMejorIndividuo($poblacion$)

Como puede observarse, cualquier algoritmo de búsqueda local podría ser incluido en el modelo propuesto. Para este estudio se van a comparar las técnicas de búsqueda en escalada y enfriamiento simulado. Dada la sensibilidad de esta última a sus parámetros, se han considerado, a su vez, dos mecanismos de aceptación clásicos, la función Metropolis y la función logística, mientras que para el decremento de la temperatura se aplica el factor de enfriamiento geométrico [12].

3.4. Algoritmo con búsqueda local como post-procesado

El Algoritmo 2 muestra el pseudocódigo para el segundo modelo propuesto, denominado EA+LS, donde la búsqueda local se realiza una vez que ha concluido la ejecución del algoritmo evolutivo. Como puede observarse, este algoritmo requiere configurar el porcentaje de la población final sobre el que se va aplicar la búsqueda ($porcentajeLS$) y el número de iteraciones deseado ($nIterLS$). Ambos parámetros se utilizan para determinar el número total de iteraciones, $totalIterLS$, que ejecutará la búsqueda local (línea 2), el cual debe ser descontado del número máximo de evaluaciones que iba a consumir el proceso evolutivo (línea 3). A continuación, se ejecuta el algoritmo evolutivo original (línea 4).

Algoritmo 2 Pseudocódigo del modelo memético EA+LS

Require: $maxEval$, $tamPobl$, $nIterLS$, $porcentajeLS$
Ensure: $mejorIndividuo$
1: $nSolucionesLS \leftarrow tamPobl \cdot (porcentajeLS/100)$
2: $totalIterLS \leftarrow nIterLS \cdot nSolucionesLS$
3: $maxEval \leftarrow maxEval - totalIterLS$
4: $poblacion \leftarrow$ evolucion($tamPobl$,$maxEval$)
5: $solucionesLS \leftarrow$ seleccionarMejores($poblacion$, $nSolucionesLS$)
6: **for all** $individuo \in solucionesLS$ **do**
7: $individuo \leftarrow$ busquedaLocal($individuo$, $nIterLS$, $poblacion$)
8: **end for**
9: asignarRankings($poblacion \cup solucionesLS$)
10: $poblacion \leftarrow$ seleccionarMejores($poblacion \cup solucionesLS$, $tamPobl$)
11: **return** seleccionarMejorIndividuo($poblacion$)

Una vez que la evolución ha concluido, las mejores soluciones de la población actual son extraídas (línea 5) de acuerdo al porcentaje establecido (línea 1). Comienza entonces el procedimiento de búsqueda local con las características detalladas en la Sección 3.3 (líneas 6-8), si bien el número de iteraciones ejecutadas es el mismo para todas las soluciones. Finalmente, la población es actualizada para incorporar las nuevas soluciones encontradas por la búsqueda local (líneas 9-10), garantizando así la devolución del mejor individuo de la población final.

4. Estudio experimental

Los modelos meméticos propuestos han sido implementados con la librería JCLEC [13], mientras que para el procesado de datos se ha utilizado la librería Datapro4j[1]. La experimentación se ha realizado en una máquina con sistema operativo Ubuntu 14.10, una CPU de 8 núcleos a 2.67 GHz y 15.6 MB de memoria RAM. Cada algoritmo se ha ejecutado 30 veces con semillas aleatorias diferentes.

El primer experimento planteado, que se corresponde con la Sección 4.2, tiene como objetivo el estudio de la eficiencia de la búsqueda local y de la influencia de sus parámetros. El experimento cuyos resultados se presentan en la Sección 4.3 analiza el comportamiento de los algoritmos meméticos desde el punto de vista de las medidas software a optimizar y la instancia del problema bajo estudio.

4.1. Configuración de parámetros

La Tabla 2 muestra los parámetros requeridos por cada algoritmo y los valores considerados en este estudio. Los parámetros comunes se han fijado en base al algoritmo original [5], mientras que para aquellos específicos de los modelos meméticos se ha definido un rango de valores amplio a fin de estudiar su influencia. Con la configuración propuesta se obtienen 27 variantes de cada modelo

[1] `http://www.uco.es/grupos/kdis/datapro4j`

Tabla 2: Configuración de parámetros empleada en la experimentación

Parámetro	**Valor**
Tamaño de la población	150
Número máximo de evaluaciones	24000
Pesos del operador de mutación	w={0.2, 0.1, 0.1, 0.3, 0.3}
Pesos de la medida ERP	$w_{as} = 2$, $w_{ag} = 3$, $w_{co} = 3$, $w_{ge} = 5$
Umbral de la medida CS	30 % del total de clases
Coeficientes de la medida CB	$\gamma = 2$, $\mu = 5$, $\omega = 8$
Probablidad LS en EA(LS)	0.01, 0.05, 0.10, 0.20, 0.80, 0.90, 0.95, 0.99, 1.00
Número de iteraciones en EA(LS)	$minIterLS = 10$, $maxIterLS = 50$, k =0.0005
Porcentaje de soluciones en EA+LS	10 %, 15 %, 20 %
Número de iteraciones en EA+LS	50, 100, 200
Temperatura inicial en SA	t=650 (Metrópolis), t=100000 (logística)
Factor de enfriamiento en SA	α =0.95

Tabla 3: Características de las instancias del problema

Problema	**#Clases**	**#Relaciones**					**#Interfaces**
		Asoc.	Depen.	Agreg.	Compos.	Gener.	
Dataprо4j	59	3	4	3	2	49	12
JSapar	46	7	33	21	9	19	80
Marvin	32	5	11	22	5	8	28
NekoHTML	47	6	17	15	18	17	46

memético, ya que a su vez se consideran tres técnicas de búsqueda local: escalada (HC), enfriamiento simulado con función Metrópolis (SA-m) y con función logística (SA-l). Para SA, la temperatura inicial está fijada de forma tal que en el estado inicial se acepte un reemplazo de la mejor solución posible por la peor (en términos de ranking) con una probabilidad igual a 0.5.

Finalmente, la Tabla 3 recoge la información sobre las instancias del problema utilizadas. Todas representan sistemas software reales, si bien difieren en el número de clases, el número de relaciones (separadas por los tipos definidos en UML 2) y el número de interfaces candidatas (relaciones unidireccionales).

4.2. Análisis de la eficiencia de la búsqueda local

Con el fin de analizar cómo de efectiva ha resultado la inclusión de la búsqueda local en el algoritmo evolutivo, las Tablas 4 y 5 muestran el porcentaje de movimientos satisfactorios y el número de evaluaciones necesarias para encontrar al mejor individuo para los modelos EA(LS) y EA+LS, respectivamente.

Como puede observarse, para el modelo EA(LS) el porcentaje de movimientos satisfactorios varía entre un 16,70 % y un 36,60 % en función del operador

Tabla 4: Eficiencia de la búsqueda local en el modelo EA(LS)

Algoritmo	**% Movimientos**	**Nº Evaluaciones**
EA	-	$23615,90 \pm 729,20$
EA(HC)_0.001	$35,60 \pm 3,20$	$23589,80 \pm 661,10$
EA(HC)_0.05	$21,60 \pm 2,90$	$23246,60 \pm 1132,40$
EA(HC)_0.10	$21,30 \pm 2,50$	$23067,20 \pm 1473,70$
EA(HC)_0.20	$20,80 \pm 2,90$	$22462,40 \pm 2482,90$
EA(HC)_0.50	$20,10 \pm 2,80$	$22421,80 \pm 2421,00$
EA(HC)_0.80	$19,50 \pm 3,00$	$22284,00 \pm 2802,00$
EA(HC)_0.90	$19,47 \pm 3,00$	$22535,70 \pm 2706,90$
EA(HC)_0.95	$19,50 \pm 3,10$	$21960,60 \pm 3025,40$
EA(HC)_1.00	$19,40 \pm 3,00$	$22460,60 \pm 2789,60$
EA(SA-m)_0.001	$35,20 \pm 3,00$	$23616,00 \pm 674,20$
EA(SA-m)_0.05	$21,90 \pm 2,70$	$23272,10 \pm 1107,80$
EA(SA-m)_0.10	$21,80 \pm 2,80$	$23111,30 \pm 1334,60$
EA(SA-m)_0.20	$22,00 \pm 2,50$	$22699,50 \pm 1889,50$
EA(SA-m)_0.50	$26,50 \pm 2,50$	$22617,00 \pm 2062,00$
EA(SA-m)_0.80	$32,90 \pm 1,90$	$22545,50 \pm 3941,70$
EA(SA-m)_0.90	$34,80 \pm 2,00$	$19079,10 \pm 7421,90$
EA(SA-m)_0.95	$35,76 \pm 2,00$	$19490,70 \pm 6870,50$
EA(SA-m)_1.00	$36,70 \pm 2,00$	$19876,60 \pm 6816,70$
EA(SA-l)_0.001	$31,90 \pm 3,00$	$23707,30 \pm 585,10$
EA(SA-l)_0.05	$17,70 \pm 2,00$	$23187,30 \pm 1389,00$
EA(SA-l)_0.10	$16,90 \pm 2,00$	$22877,00 \pm 1551,40$
EA(SA-l)_0.20	$16,70 \pm 1,90$	$22803,50 \pm 1657,00$
EA(SA-l)_0.50	$18,70 \pm 1,50$	$19013,50 \pm 6450,20$
EA(SA-l)_0.80	$19,30 \pm 1,70$	$14709,00 \pm 7221,50$
EA(SA-l)_0.90	$19,67 \pm 1,70$	$14986,50 \pm 7278,00$
EA(SA-l)_0.95	$19,60 \pm 1,60$	$13019,30 \pm 6749,80$
EA(SA-l)_1.00	$19,70 \pm 1,60$	$15455,00 \pm 7143,30$

concreto considerado y su probabilidad de ejecución[2]. Sin embargo, este modelo encuentra la solución que será devuelta al final como la mejor en un número menor de evaluaciones que el algoritmo evolutivo original (EA). Un aspecto a destacar es que el incremento de la probabilidad no suele tener un efecto importante en la eficiencia de los movimientos para un mismo operador, si bien sí se aprecian mayores diferencias en relación al número de evaluaciones. Por lo general, probabilidades altas implican acelerar más la obtención del mejor individuo, aunque también supone una mayor variación entre las distintas ejecuciones.

En el caso del modelo EA+LS, los resultados difieren bastante de los anteriores. Por un lado, la eficiencia de la búsqueda local parece ser mayor aquí, pero esto no se refleja en la aceleración del proceso de búsqueda completo. Co-

[2] Por motivos de espacio, y dado que no hay grandes diferencias entre las instancias del problema según la desviación estándar, sólo se muestran los resultados globales.

Tabla 5: Eficiencia de la búsqueda local en el modelo EA+LS

Algoritmo	**% Movimientos**	**Nº Evaluaciones**
EA	-	$23615,90 \pm 729,20$
EA+HC_10%_50	$100,00 \pm 0,00$	$23347,50 \pm 55,70$
EA+HC_10%_100	$100,00 \pm 0,00$	$22598,40 \pm 55,70$
EA+HC_10%_200	$100,00 \pm 0,00$	$21096,80 \pm 56,70$
EA+HC_15%_50	$100,00 \pm 0,00$	$22945,50 \pm 29,40$
EA+HC_15%_100	$100,00 \pm 0,00$	$21883,30 \pm 23,20$
EA+HC_15%_200	$100,00 \pm 0,00$	$19647,30 \pm 30,00$
EA+HC_20%_50	$100,00 \pm 0,00$	$22598,40 \pm 55,70$
EA+HC_20%_100	$100,00 \pm 0,00$	$21096,80 \pm 56,70$
EA+HC_20%_200	$100,00 \pm 0,00$	$18099,00 \pm 57,00$
EA+SA-m_10%_50	$100,00 \pm 0,00$	$23347,50 \pm 55,70$
EA+SA-m_10%_100	$100,00 \pm 0,00$	$22598,40 \pm 55,70$
EA+SA-m_10%_200	$100,00 \pm 0,00$	$21096,80 \pm 56,70$
EA+SA-m_15%_50	$100,00 \pm 0,00$	$22945,50 \pm 29,40$
EA+SA-m_15%_100	$100,00 \pm 0,00$	$21883,30 \pm 23,20$
EA+SA-m_15%_200	$100,00 \pm 0,00$	$19647,30 \pm 30,00$
EA+SA-m_20%_50	$100,00 \pm 0,00$	$22598,00 \pm 55,70$
EA+SA-m_20%_100	$100,00 \pm 0,00$	$21096,80 \pm 56,70$
EA+SA-m_20%_200	$100,00 \pm 0,00$	$18099,00 \pm 57,00$
EA+SA-l_10%_50	$50,00 \pm 1,80$	$23347,50 \pm 55,70$
EA+SA-l_10%_100	$49,80 \pm 1,20$	$22598,40 \pm 55,70$
EA+SA-l_10%_200	$49,90 \pm 0,90$	$21096,80 \pm 56,70$
EA+SA-l_15%_50	$50,03 \pm 1,40$	$22945,50 \pm 29,40$
EA+SA-l_15%_100	$49,82 \pm 1,10$	$21883,30 \pm 23,20$
EA+SA-l_15%_200	$49,93 \pm 0,80$	$19647,30 \pm 30,00$
EA+SA-l_20%_50	$49,90 \pm 1,30$	$22598,40 \pm 55,70$
EA+SA-l_20%_100	$49,90 \pm 0,90$	$21096,80 \pm 56,70$
EA+SA-l_20%_200	$50,00 \pm 0,60$	$18099,00 \pm 55,70$

mo puede observarse, el número de evaluaciones empleado es el mismo para los tres operadores y cada una de las combinaciones consideradas. Esto se debe a que la solución devuelta como mejor individuo fue encontrada por el algoritmo evolutivo, por lo que la búsqueda local, a pesar de explorar de manera efectiva el entorno de vecindad, no consigue encontrar soluciones mejores. Con el fin de establecer otras direcciones de búsqueda, este modelo memético acepta como satisfactorios aquellos movimientos a soluciones vecinas que, teniendo el mismo *fitness* (ranking en la población), representan soluciones diferentes a la inicial. Esta circunstancia afecta claramente al resultado del algoritmo, puesto que aunque la población final mejora con respecto a la que se obtuvo tras la fase de evolución, este hecho no repercute en la solución final devuelta al arquitecto. Por tanto, la aplicación del modelo EA+LS no resulta efectiva para resolver el problema del descubrimiento de arquitecturas.

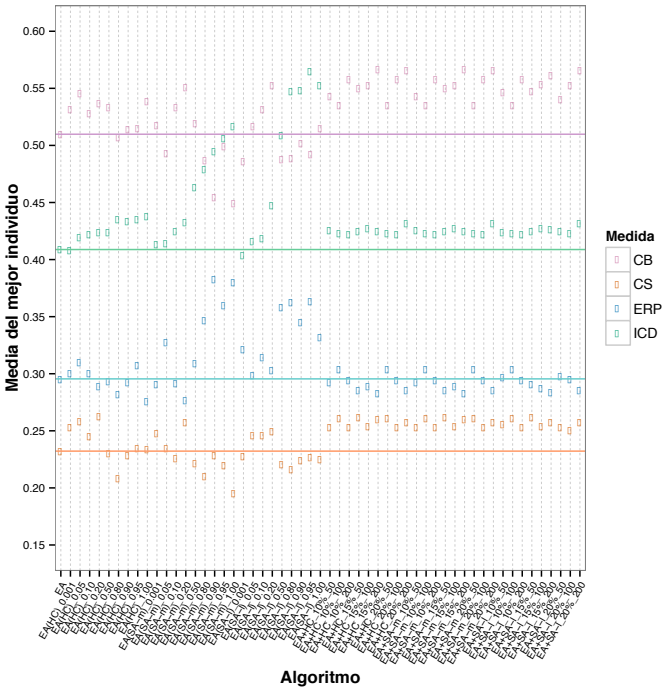


Figura 1: Valores medios de las medidas software para la instancia *JSapar*

4.3. Análisis de la optimización de medidas software

Con el fin de comprender la influencia de los parámetros en las soluciones obtenidas, aquí se analiza la optimización alcanzada para cada una de las medidas software. En este sentido, los datos muestran cierta variabilidad en función de la instancia del problema y la medida software considerada.

Para *Datapro4j*, la mayor optimización se consigue para la medida CB, especialmente cuando se emplea el modelo EA(HC) con probabilidades altas. Sin embargo, este tipo de algoritmo provoca el efecto contrario para otras medidas como CS y, particularmente, ERP. En este sentido, el algoritmo original mantiene un mejor balance entre medidas. A pesar de ser el sistema con más clases,

el número de interfaces candidatas es muy inferior al resto de problemas, por lo que se trata de un problema para el que se presuponía poco margen de mejora.

Para instancias más complejas, como *JSapar*, se aprecian más diferencias en función del operador aplicado. La técnica de enfriamiento simulado, con cualquiera de los dos mecanismos de aceptación, obtiene mejoras para las medidas CB y CS cuando se utiliza el modelo EA(LS), pero no consigue mantener un balance adecuado con las otras dos medidas. Por otro lado, la búsqueda en escalada consigue optimizaciones puntuales para las medidas ERP, CS y CB sin alterar demasiado los valores de ICD. A modo de ejemplo, la Figura 1 muestra el valor medio alcanzado por el mejor individuo devuelto por el algoritmo correspondiente para cada una de las medidas software consideradas. Por motivos de claridad, los valores se presentan escalados en un rango [0,1], donde el óptimo es 0 para todas las medidas.

El modelo EA+LS únicamente aporta resultados interesantes para la instancia *Marvin*, especialmente cuando se aplica la técnica SA-m. En general, este modelo parece ser más robusto que el modelo EA(LS) frente a la configuración de parámetros elegida, si bien su rendimiento raramente mejora al algoritmo evolutivo original. Finalmente, el comportamiento de los modelos meméticos para *NekoHTML* se asemeja al observado para *Datapro4j*, aunque la primera instancia tiene una mayor complejidad debido al número de interfaces candidatas. En esta ocasión, las medidas para las que se consigue mejora son ERP y CS, mientras que ICD es la medida más perjudicada.

5. Conclusiones

Automatizar tareas propias del análisis arquitectónico mediante técnicas metaheurísticas plantea retos no sólo a la hora de su formulación como problemas de optimización, sino también de adecuar la elección de los algoritmos de búsqueda a sus características. Aunque la combinación de métodos a priori complementarios como los algoritmos evolutivos y la búsqueda local puede ayudar a mejorar el rendimiento del proceso de optimización, el desarrollo de mecanismos de hibridación efectivos requiere de un gran esfuerzo, pues existen multitud de decisiones de diseño que afectan en gran medida al rendimiento del algoritmo resultante.

Es por ello que aquí se ha planteado un estudio comparativo de dos modelos meméticos con los que abordar el descubrimiento evolutivo de arquitecturas software. La experimentación realizada ha permitido comprobar que la búsqueda local, aplicada como operador genético, ayuda a obtener buenas soluciones en un número menor de evaluaciones y que también es posible mejorar la calidad de las soluciones respecto a algunas de las medidas software consideradas. Sin embargo, esto supone a menudo un deterioro en otras medidas, aspecto que el ingeniero debería valorar a la hora de aplicar un algoritmo u otro. Por el contrario, la aplicación de la búsqueda local tras la evolución no ha resultado efectiva, ya que la búsqueda global converge a soluciones donde las medidas software están mejor balanceadas.

Como trabajo futuro se prevé extender este análisis a otras combinaciones de medidas y otras instancias, a fin de obtener conclusiones más generalizables respecto a las dependencias vislumbradas aquí. Así mismo, abordar la integración de la búsqueda local en algoritmos evolutivos multi-objetivo constituye una línea de trabajo interesante.

Agradecimientos

Trabajo financiado por el Ministerio de Economía y Competitivad, proyecto TIN2014-55252-P y el Ministerio de Educación, programa FPU (FPU13/01466).

Referencias

1. M. Harman, S. Afshin Mansouri, and Y. Zhang, "Search Based Software Engineering: Trends, Techniques and Applications," *ACM Computing Surveys*, vol. 45, no. 1, pp. 1–64, 2012.
2. I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Information Sciences*, vol. 237, pp. 82–117, 2013.
3. O. Räihä, "A survey on search-based software design," *Computer Science Review*, vol. 4, no. 4, pp. 203–249, 2010.
4. A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software Architecture Optimization Methods: A Systematic Literature Review," *IEEE Transactions on Software Engineering*, vol. 39, no. 5, pp. 658–683, 2013.
5. A. Ramírez, J. R. Romero, and S. Ventura, "An approach for the evolutionary discovery of software architectures," *Information Sciences*, vol. 305, pp. 234–255, 2015.
6. C. Blum, J. Puchinger, G. R. Raidl, and A. Roli, "Hybrid Metaheuristics in Combinatorial Optimization: A Survey," *Applied Soft Computing*, vol. 11, no. 6, pp. 4135–4151, 2011.
7. N. Krasnogor and J. Smith, "A Tutorial for Competent Memetic Algorithms: Model, Taxonomy, and Design Issues," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 5, pp. 474–488, 2005.
8. M. Lozano and C. García-Martínez, "Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report," *Computers and Operations Research*, vol. 37, no. 3, pp. 481–497, 2010.
9. J. Smith and C. L. Simons, "A comparison of two memetic algorithms for software class modelling," in *Proc. 15th Annual Conference on Genetic and evolutionary computation (GECCO'13)*, p. 1485, 2013.
10. O. Sievi-Korte, M. Erkki, and T. Poranen, "Simulated Annealing for Aiding Genetic Algorithm in Software Architecture Synthesis," *Acta Cybernetica*, vol. 21, pp. 235–265, 2013.
11. N. K. Bambha, S. S. Bhattacharyya, J. Teich, and E. Zitzler, "Systematic Integration of Parameterized Local Search Techniques in Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 2, pp. 137–155, 2004.
12. E. Aarts and J. Korst, *Essays and Surveys in Metaheuristics*, ch. Selected Topics in Simulated Annealing, pp. 1–37. Springer US, 2002.
13. S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervás, "JCLEC: A Java framework for evolutionary computation," *Soft Computing*, vol. 12, no. 4, pp. 381–392, 2008.

Providing Support for the Optimized Management of Declarative Processes

Irene Barba[1], Andreas Lanz[2] Andrés Jiménez-Ramírez[1], Barbara Weber[3], Manfred Reichert[2], and Carmelo Del Valle[1]

[1] Departamento de Lenguajes y Sistemas Informáticos, University of Seville, Spain,
{irenebr,ajramirez,carmelo}@us.es
[2] Institute of Databases and Information Systems, Ulm University, Germany
{Andreas.Lanz,Manfred.Reichert}@uni-ulm.de
[3] Department of Computer Science, University of Innsbruck, Austria,
Technical University of Denmark, Denmark
bweb@dtu.dk

Abstract. Declarative process models are becoming increasingly popular due to the high flexibility they offer to process participants. Based on a declarative process model, there exist numerous possible enactment plans, each one with specific values for relevant objective functions (e.g., overall completion time). How to actually execute such a model is quite challenging due to several reasons: (1) proper objective functions must be considered to find optimized enactment plans, (2) users often do not have an understanding of the overall process, (3) the presence of a variety of temporal constraints to be met during process enactment, and (4) the need to coordinate multiple instances of a process concurrently executed (which compete for shared resources). This is further complicated by the fact that the enactment of new process instances may continuously start over time and many organizations do not exactly know their future demands. In such context, to properly support users in enacting declarative process models, this paper suggests generating optimized enactment plans from declarative process models. The generated enactment plans may be used for different purposes, e.g., to provide personal schedules to users. Moreover, they may be dynamically adapted if required. To evaluate the applicability of our approach in practical settings we apply it to a real process scenario from the healthcare domain.

Keywords: Process flexibility, declarative process model, temporal constraints, constraint programming, scheduling, healthcare processes

1 Introduction

For several years, there has been an increasing interest in aligning information systems in a process-oriented way [2, 15]. Usually, business processes (BPs) have numerous constraints to be obeyed during process enactment. To provide operational support, BPs are mostly specified in an imperative way, i.e., defining a precise schema that establishes how a given set of activities has to be performed. However, imperative process models are often too rigid to meet the flexibility requirements of users. As an alternative, therefore, declarative process models are increasingly used [13, 9] since they provide an

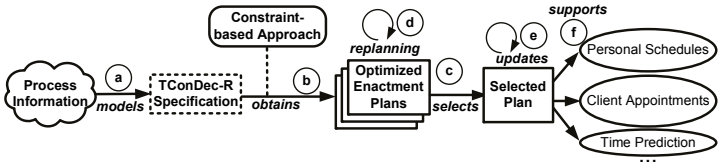


Fig. 1. Overview of our approach.

increased flexibility when executing the process, allowing users to specify *what* has to be done instead of *how* [8].

On one hand, a declarative process model offers a high flexibility to end users by allowing for the required degree of freedom. On the other, executing a declarative model usually entails larger efforts for users compared to imperative models [10]. In general, numerous enactment plans related to the same declarative model exist, each one presenting specific values for relevant objective functions (e.g., overall completion time). Consequently, the decision on how to execute a declarative model is quite complex. It becomes even more challenging when dealing with business processes that (1) contain complex temporal as well as cross-instance constraints, (2) require an efficient management of shared resources, and (3) need to consider the optimization of specific objective functions in an uncertain and changing environment. In such a context, usually, users neither know the complete BP nor the impact of their actions. Both might result in suboptimal enactment plans. Therefore, advanced user support is required when executing declarative process models. Such support can be provided by automatically generating optimized enactment plans from the respective declarative process model. Moreover, during enactment it should be possible to flexibly adapt these plans if required taking relevant run-time information into account.

Optimized enactment plans foster sophisticated user support through: (1) personal schedules [4], i.e., assisting process participants by recommending which activities they shall perform when, (2) reduced turnaround and waiting times, i.e., organizing appointments while taking the start times of the activities the users are involved in into account, and (3) time predictions, i.e., predicting enactment times for future activities. Note that (1) allows for a better planning of work, while (3) enables decision support for process participants [14].

2 Contribution

This paper suggests the generation of optimized enactment plans for a set of process instances derived from a declarative model. Figure 1 provides an overview of our approach. First, the declarative specification of the BP is defined (Step a in Fig. 1). For this, we consider the declarative language DECLARE[1] [13] as a basis. In order to address the described problems, DECLARE is extended by (1) including capabilities for reasoning about resources as well as the estimated duration of process activities, and

[1] DECLARE is one of the most referenced and used declarative process modeling languages.

(2) supporting common time patterns reported in literature [6, 5] (e.g., cross-instance constraints related to time). This results in the TConDec-R language.

From a TConDec-R specification, in turn, optimized enactment plans can be automatically generated (Step b in Fig. 1). In this context, the activities to be executed are selected and ordered (i.e., *planning problem* [3]), considering control-flow constraints as well as temporal and resource constraints imposed by the declarative specification (i.e., *scheduling problem* [1]). For planning and scheduling the activities in a way such that a given objective function becomes optimized (i.e., for generating a set of optimized enactment plans), a constraint-based approach can be proposed. Finally, from the set of optimized enactment plans (i.e., schedules), the plan that fits best to the scenario requirements is selected for enactment (Step c in Fig. 1). The selected plan is then used for improving process support in several respects (Step f in Fig. 1).

Additionally, the proposed approach supports the dynamic adaptation of optimized enactment plans during run-time through replanning, and hence enables an increased run-time flexibility (Step d in Fig. 1). Each time the set of optimized enactment plans is updated, a new plan for providing process support is selected. In general, replanning might become necessary either if the actual enactment of the process instances deviates from the optimized enactment plan generated earlier (e.g., estimates might turn out to be inaccurate or resource availability might unexpectedly change) or in scenarios in which the enactment of new process instances continuously starts and the demand of the services to be provided varies over time. Respective scenarios are known as lot-sizing ones with uncertain demand [12].

To evaluate the applicability of our approach in practical settings we consider to apply it to a real process scenario from the healthcare domain. The considered scenario deals with the scheduling of surgeries and their preparations in the context of ovarian carcinoma [11, 7]. The application of the proposed approach to the medical scenario offers several advantages: (1) avoiding constraint violations, (2) managing shared resources in an effective way, (3) improving and automating (inter-process) coordination between different hospitals, (4) reducing waiting times of patients as the schedule becomes known beforehand, and (5) minimizing the length of patient stays in hospitals.

Note that the approach is not restricted to healthcare environments, i.e., it can be also applied in other domains for which process flexibility and temporal constraints play a crucial role (e.g., automotive engineering and flight planning [6]).

The proposed approach combines original contributions regarding (1) the formal specification and operational support of complex temporal constraints during process enactment, (2) cross-instance coordination, (3) resource allocation (i.e., scheduling) before and during process enactment, and (4) optimization of objective functions.

3 Conclusion

This paper proposed the generation of optimized enactment plans (e.g., minimizing overall completion time) from declarative temporal process models. This generation is quite challenging since it considers complex temporal constraints, cross-instance coordination, resource allocation before and during process enactment, and optimization of a given objective function. It is further complicated by the fact that the enactment of

new process instances may continuously start over time and many organizations do not exactly know their future demands.

The generated plans can be used for different purposes, e.g., providing users with a personal schedule, suggesting appointments to process stakeholders, or predicting enactment times for process activities. Moreover, these plans are adapted during the enactment of the process if necessary.

As future work, we will test our approach in the context of ovarian carcinoma.

Acknowledgment

This work has been partially funded by the Spanish Ministerio de Economía y Competitividad (TIN2013-46928-C3-3-R and TIN2015-71938-REDT).

References

1. P. Brucker and S. Knust. *Complex Scheduling (GOR-Publications)*. Springer, 2006.
2. M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software through Process Technology*. Wiley-Interscience, Hoboken, NJ, 2005.
3. M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, Amsterdam, 2004.
4. A. Lanz, J. Kolb, and M. Reichert. Enabling personalized process schedules with time-aware process views. In *CAiSE'13 Workshops*, LNBIP, pages 205–216. Springer, 2013.
5. A. Lanz, M. Reichert, and B. Weber. Process time patterns: A formal foundation. *Information Systems*, 57:38–68, 2016.
6. A. Lanz, B. Weber, and M. Reichert. Time patterns for process-aware information systems. *Requirements Engineering*, 19(2):113–141, 2014.
7. Ovarian cancer (CG122). `http://www.nice.org.uk/CG122`, 2011. [Online; accessed 7-April-2016].
8. M. Pesic. *Constraint-Based Workflow Management Systems: Shifting Control to Users*. PhD thesis, Technische Universiteit Eindhoven, The Netherlands, 2008.
9. P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, and H.A. Reijers. Imperative versus Declarative Process Modeling Languages: An Empirical Investigation. In *Proc. BPM Workshops*, pages 383–394, 2011.
10. M. Reichert and B. Weber. *Enabling Flexibility in Process-Aware Information Systems*. Springer, 2012.
11. B. Schultheiß, J. Meyer, R. Mangold, T. Zemmler, and M. Reichert. Designing the processes for ovarian cancer surgery (in german). Technical Report DBIS-6, University of Ulm, 1996.
12. L. Tiacci and S. Saetta. Demand forecasting, lot sizing and scheduling on a rolling horizon basis. *International Journal of Production Economics*, 140(2):803–814, 2012.
13. W.M.P. van der Aalst, M. Pesic, and M.H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23(2):99–113, 2009.
14. W.M.P. van der Aalst, M.H. Schonenberg, and M. Song. Time prediction based on process mining. *Information Systems*, 36(2):450–475, 2011.
15. M. Weske. *Business Process Management: Concepts, Languages, Architectures (Second Edition)*. Springer, 2012.

Configuración guiada por búsqueda de aplicaciones basadas en microservicios en la nube

José Antonio Parejo[1*], Aurora Ramírez[2], José Raúl Romero[2], Sergio Segura[1] y Antonio Ruiz-Cortés[1]

Dpto. de Lenguajes y Sistemas Informáticos, Univ. de Sevilla[1],
Dpto. de Informática y Análisis Numérico, Universidad de Córdoba[2]
japarejo@us.es,{aramirez,jrromero}@uco.es,{sergiosegura,aruiz}@us.es

Resumen Organizaciones como Netflix, Google o Amazon hacen uso de arquitecturas basadas en microservicios, lo que ha disparado el interés de la comunidad en ingeniería del software por este estilo arquitectónico. No obstante, el buen uso de este estilo arquitectónico supone nuevos retos, como determinar qué instancias de servicios se despliegan o establecer la mejor configuración de la nube que los aloja, conforme a la carga de trabajo esperada para cumplir los Acuerdos de Nivel de Servicio. Se trata de un problema de optimización en el que deben considerarse simultáneamente múltiples propiedades, a menudo en conflicto entre sí. Por ello, tras formular este caso como un problema de búsqueda, se discutirá cómo el uso de técnicas multi-objetivo puede mejorar las soluciones actuales, permitiéndonos escoger los proveedores y configuraciones apropiadas para disminuir los costes de explotación, y asegurar la disponibilidad de los servicios críticos sin empobrecer el tiempo de respuesta.

Keywords: microservicios, optimización multi-objetivo, *cloud computing*, Acuerdos de Nivel de Servicio

1. Introducción

La evolución de las arquitecturas distribuidas ha desembocado en la definición de un estilo arquitectónico denominado arquitectura basada en microservicios. Este estilo arquitectónico ha sido aplicado con éxito en organizaciones como Google, Ebay [5], Netflix [4], o Amazon [2] para aplicaciones con cargas de peticiones masivas que deben ofrecer tiempos de respuesta bajos, y alta disponibilidad.

En este tipo de aplicaciones los distintos módulos o historias de usuario de la aplicación se implementan como servicios web RESTful independientes. De esta forma, se alcanza un nivel de modularidad que facilita el control del redespliegue en únicamente aquellas partes que soportan mayor carga de trabajo y, consecuentemente, evitan el uso indiscriminado de la infraestructura.

[*] Este trabajo ha sido sufragado parcialmente por la Comisión Europea(FEDER), y los gobiernos de España y Andalucía mediante los proyectos BELI (TIN2015-70560-R), THEOS (TIC-5906), COPAS (TIC-1867) y TIN2014-55252-P, la Red de Excelencia SEBASENet (TIN2015-71841-REDT) y el programa FPU (FPU13/01466).

Sin embargo, diversos autores han señalado que, junto con las ventajas que ofrece este estilo arquitectónico, aparecen también nuevos retos y decisiones a tomar. Una de ellas es determinar la manera en que los microservicios se distribuirán en las distintas máquinas virtuales (MVs) en el caso de realizar un despliegue en la nube. Concretamente, con el objetivo de aprovechar la elasticidad y el modelo de pago por uso que se emplea normalmente en la nube se hace necesario decidir: i) las MVs que se van a usar y sus características concretas, ii) cuántas instancias de cada microservicio se van a desplegar, y en qué maquinas virtuales concretas lo harán, y iii) las reglas de escalado a aplicar sobre la infraestructura y los microservicios para adaptarse a la carga de trabajo conforme a los Acuerdos de Nivel de Servicio (ANS) establecidos. En este artículo, se presenta y formaliza este problema, identificando un conjunto de objetivos de interés que lo configuran como un problema multi-objetivo susceptible de ser abordado mediante técnicas de búsqueda.

2. Ejemplo motivador

Supongamos una aplicación de gestión de personal donde se controla el horario, acceso a las instalacciones, y las tareas que realizan los empleados de una organización durante su jornada de trabajo. Dichas tareas corresponden además a diversos proyectos que la organización desea gestionar. Podríamos identificar 4 microservicios: gestión de usuarios, gestión de instalaciones y control de acceso, gestión de tareas y proyectos, y gestión de sesiones de trabajo (que asocian a usuarios con tareas concretas durante un número de horas). Cada microservicio podría tener asociado una base de datos (BD) para la gestión de su información, que es compartida por todas sus instancias, y se tendrá adicionalmente un registro de instancias de servicios disponibles que es a su vez otro microservicio. El esquema de despliegue más simple posible, es aquel en el que todos los microservicios y bases de datos se despliegan sobre la misma MV, y no se realiza ningún tipo de escalado en base a la utilización. Uno de los esquemas de despliegue alternativo posibles es aquel en el que hay 2 MVs, y cada microservicio tiene un único despliegue en alguna de las MVs. La figura 1 muestra ambos esquemas como diagramas de componentes UML decorados.

3. Configuración de aplicaciones basadas en microservicios en la nube

Sea $S = \{s_1, \ldots, s_n\}$ el conjunto de microservicios a desplegar en la nube. Sea $N = \{n_1, \ldots, n_{MaxMV}\}$ un conjunto de máquinas virtuales, cada una con una configuración de proveedor de cloud determinada. Sea $C = \{c_1, \ldots, c_l\}$ el conjunto de configuraciones posibles de máquinas virtuales proporcionadas por un conjunto de proveedores de infraestructura en la nube. Una configuración de la arquitectura cloud viene determinada por el valor de las variables: $x_{i,j}$, $y_{i,j}$, y z_j, ST y GT. Donde $x_{i,j}$ son variables binarias que indican si el microservicio

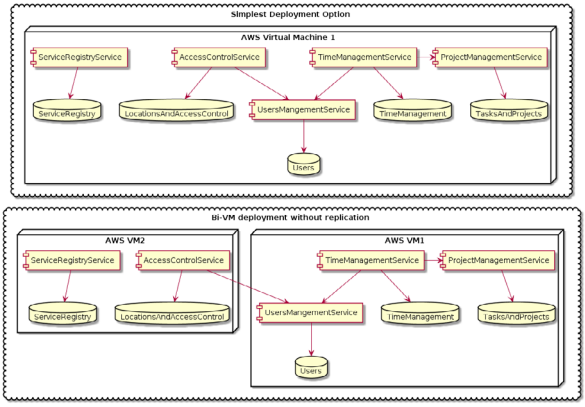


Figura 1. Dos opciones de despliegue para nuestro ejemplo

i está despegado en la MV j, para cada valor de i, al menos una variable tiene un valor de 1, es decir, todo microservicio se despliega en al menos una MV. Las variables binarias $y_{i,j}$ indican si la BD asociada al microservicio i está desplegada en la MV j, para cada i, solo una variable tiene un valor de 1, es decir, la BD de cada servicio se despliega en una única MV. Las variables $z_j \in C$ indican la configuración concreta de la MV j. GT indica el límite de utilización de las MVs a partir del cual se aplicará una regla de crecimiento de la infraestructura (se instanciará una nueva VM y se desplegarán nuevas instancias de los microservicios con más carta), y ST que indica el límite de utilización de las MVs a partir del cual se aplicará una regla de reducción de la infraestructura (se destruirá una VM). Obviamente, $0 \leq ST < GT \leq 100$, donde 100 significa una utilización máxima de la potencia de computación de las VMs instanciadas actualmente. $MaxMV$ es el límite máximo de MVs que se pueden desplegar para una configuración.

Dado un conjunto A de ANSs acordados con distintos usuarios, y una carga de trabajo esperada ω para un periodo de tiempo concreto (que especifica el volumen de peticiones que se realizarán en cada instante para cada servicio y el ANS concreto que corresponde a cada una), un conjunto interesante de objetivos a optimizar para una instancia de este problema sería:

- **Minimización del coste total de la infraestructura** $D(\omega, x, y, z)$. Este objetivo pretende minimizar el coste total del despliegue calculado como el sumatorio del coste de cada MV, que depende a su vez de la configuración de dichas VMs.

- **Minimización del número de puntos únicos de fallo** $F(x, y, z)$. Si un microservicio no está replicado, ese único despliegue supone un punto único de fallo para la lógica de la aplicación. Este objetivo pretende minimizar el número de puntos de este tipo sobre todo el conjunto de microservicios, para hacer la aplicación lo más robusta posible.
- **Minimización del tiempo de respuesta medio** $T(\omega, x, y, z)$. Dependiendo del número de despliegues, la configuración de las MVs y el perfil de carga de peticiones de cada servicio, tendremos un tiempo medio de repuesta para las peticiones. Este objetivo pretende minimizar dicho tiempo.
- **Minimización del número de violaciones de los ANS** $V(A, \omega, x, y, z)$. Dado un conjunto de ANSs con los distintos usuarios, es posible simular el comportamiento del sistema bajo un determinado perfil de carga de peticiones por usuario. Esto permitiría calcular el número de peticiones para las que se han incumplido los ANS.
- **Minimización del número de reconfiguraciones en la arquitectura** $R(A, \omega, x, y, z)$. Una reconfiguración es la instanciación o parada de una MV, con los correspondientes despliegues de servicios y BDs. Pueden existir limitaciones o costes asociado a las reconfiguraciones.

Establecer el equilibrio apropiado entre objetivos contrapuestos como el coste y el tiempo de respuesta va a depender en gran medida de las preferencias de los usuarios. Además, dado que cada MV puede tener asociado un gran número de configuraciones (más de 17.000 para Amazon [3]), el espacio de búsqueda resultante es enorme. Por ello, planteamos la necesidad de abordar este problema de optimización multi-objetivo mediante técnicas de búsqueda heurísticas. En concreto, los algoritmos evolutivos multi-objetivo (MOEAs) nos permitirán explorar de manera eficiente el espacio de configuraciones, estableciendo diferentes compromisos entre los criterios de calidad. En cuanto al trabajo futuro, pretendemos estudiar las características del problema para aplicar MOEAs, ya que puede ser necesario aplicar técnicas avanzadas dado el número de objetivos definidos. Para ello utilizaremos datos reales de Amazon, y el simulador CDOSim [1].

Referencias

1. Fittkau, F., Frey, S., Hasselbring, W.: Cdosim: Simulating cloud deployment options for software migration support. In: 6th IEEE Int. Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems. pp. 37–46 (2012)
2. Fulton, S.M.: What led amazon to its own microservices architecture. Web Page (2015), http://thenewstack.io/led-amazon-microservices-architecture/
3. García-Galán, J., Trinidad, P., Rana, O.F., Ruiz-Cortés, A.: Automated configuration support for infrastructure migration to the cloud. Future Generation Computer Systems 55, 200–212 (2016)
4. Mauro, T.: Adopting microservices at netflix: Lessons for architectural design. Blog entry, https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/
5. Shoup, R.: Service architectures at scale: Lessons from google and ebay. Talk at InfoQ.com, http://www.infoq.com/presentations/service-arch-scale-google-ebay

Minimización de conjuntos de casos de prueba en la prueba de mutaciones de composiciones BPEL

Francisco Palomo Lozano, Antonia Estero Botaro e Inmaculada Medina Bulo *

Departamento de Ingeniería Informática, Universidad de Cádiz,
Avda. de la Universidad de Cádiz 10, 11519 Puerto Real, Spain
{francisco.palomo,antonia.estero,inmaculada.medina}@uca.es

Resumen Tanto en la aplicación de prueba de mutaciones a composiciones BPEL como en la realización de estudios experimentales sobre diversas métricas de calidad, surge la necesidad de minimizar conjuntos de casos de prueba manteniendo la máxima cobertura de mutación. La prueba de este tipo de software presenta algunas peculiaridades. Normalmente, las composiciones son relativamente pequeñas cuando se comparan con aplicaciones tradicionales, pues se encargan exclusivamente de la orquestación de los servicios, y no se dispone de un gran número de casos de prueba para ellas. No obstante, su ejecución puede resultar muy costosa, y debe realizarse para un número de mutantes que, normalmente, supera ampliamente al de casos de prueba. Se propone aquí como técnica de minimización una reducción a programación lineal entera y se evalúa su rendimiento para distintas composiciones.

Keywords: Minimización de conjuntos de casos de prueba, Programación lineal entera, Prueba de mutaciones, Composiciones BPEL, SBSE.

1. Introducción

La prueba de software supone una parte importante del coste de desarrollo que las estimaciones más conservadoras elevan hasta un 40 %. Surge la necesidad de seleccionar cuidadosamente qué casos de prueba ejecutar, o en qué orden, para que se detecten errores potenciales al menor coste [9]. Existen dos situaciones en las que el coste de ejecución de las pruebas es muy alto, pero por motivos distintos. En la primera, la más común, se dispone de numerosos casos de prueba, probablemente heredados a lo largo de diferentes versiones. En la segunda, el conjunto de casos de prueba es relativamente pequeño, pero se emplea con un número elevado de características del sistema en pruebas (SUT).

Esta segunda situación es frecuente en el desarrollo de composiciones de servicios web. Las composiciones combinan servicios que se prueban por separado o cuya prueba no es responsabilidad de la organización que desarrolla o mantiene la composición. De hecho, puede que estén disponibles varios servicios y

* Parcialmente financiado por la red de excelencia TIN2015-71841-REDT (SEBASENet) y los proyectos TIN2014-60844-R (SAVANT) y TIN2015-65845-C3-3-R (DArDOS).

dinámicamente se seleccione entre ellos cuál ejecutar realmente en cada momento, dependiendo de una serie de condiciones. En conclusión, los servicios juegan para la composición el papel de cajas negras u oráculos, a los que se realiza una petición y devuelven un resultado. La labor del equipo de pruebas consiste en diseñar casos de prueba efectivos para la composición en sí. Estos pueden no ser numerosos, pero su coste de ejecución es generalmente muy alto.

No obstante, la eliminación de casos de prueba no puede abordarse efectivamente sin contar con métricas que permitan evaluar la efectividad de los casos de prueba seleccionados. Puesto que no existe un único criterio posible a la hora de fijar estas métricas e incluso pueden existir objetivos contrapuestos, es necesario especificar claramente el contexto. Presentamos aquí un enfoque para la minimización de conjuntos de casos de prueba basado en cobertura de mutaciones para composiciones WS-BPEL [7]. Para ello emplearemos técnicas exactas basadas en programación lineal entera (ILP).

2. Trabajos relacionados

Existe una vasta literatura en torno al empleo de técnicas metaheurísticas. No obstante, en aras de la brevedad, citaremos solo algunos trabajos relevantes que se han ocupado de las técnicas exactas de minimización de casos de prueba.

La mayoría de los autores que emplean técnicas exactas recurren a algún tipo de reducción de un problema de recubrimiento NP-difícil a otro problema objetivo, también NP-difícil, pero más general. Para estos problemas objetivo existen optimizadores muy eficientes que han sido desarrollados y mejorados durante décadas, en ocasiones con una fuerte inversión del sector privado.

Normalmente, el problema objetivo es ILP. Entre los trabajos en los que se aplica ILP a la minimización de casos de prueba pueden destacarse [4,2,10,8]. No obstante, un enfoque más reciente consiste en reducir la minimización a un problema SAT extendido, en lugar de a ILP, como en [1,6].

A diferencia del presente trabajo, la mayoría de autores supone un coste de ejecución idéntico para cada caso de prueba. Esto permite reducir el número de casos de prueba (por ejemplo, por subsunción) antes de abordar la minimización. Muchos usan la suite SIEMENS[1] donde la reducción tienen un gran impacto: tras aplicarla, el programa más complejo resultante es `replace`, con 215 casos de prueba y 208 características. Otro enfoque es el de [5], donde la métrica es la energía consumida durante el proceso de prueba; también emplea ILP.

3. Técnica de minimización

Aunque la técnica descrita a continuación es general y puede aplicarse a prácticamente cualquier SUT y marco de pruebas, se desarrollará específicamente para composiciones WS-BPEL empleando como marco de pruebas MuBPEL. Para una descripción de MuBPEL y de la terminología que sigue, véase [3].

[1] Disponible en http://sir.unl.edu/portal.

A partir de los registros producidos por MuBPEL para la composición se extrae la *matriz de ejecución*, E. Si se suministran los parámetros adecuados, los tiempos de ejecución de cada caso de prueba frente a cada mutante quedan registrados y puede obtenerse en paralelo la *matriz de costes* de ejecución, C. E y C son de dimensión $m \times n$, siendo $m = |M|$ y $n = |T|$.

En la matriz de ejecución, $e_{ij} = 1$ exactamente cuando m_i cubre a t_j o, equivalentemente, t_j mata a m_i. Si $e_{ij} = 2$ para algún j entonces m_i es un *mutante inválido*. En la matriz de costes, c_{ij} es el tiempo de ejecución del mutante m_i para el caso de prueba t_j. Las filas correspondientes a mutantes inválidos pueden eliminarse de E y C, por lo que puede suponerse que E es una matriz binaria y que los tiempos de prueba contenidos en C corresponden a ejecuciones válidas. También se elimina una fila si corresponde a un *mutante vivo*.

El objetivo es minimizar el número de casos de prueba en composiciones WS-BPEL empleando como métricas la *cobertura de mutaciones* y el *coste de ejecución*. En el primer escenario se debe preservar la máxima cobertura, mientras que en el segundo se debe además minimizar el coste global de ejecución de los casos de prueba seleccionados, esto es, el tiempo total de prueba. Ambos se reducen al siguiente ILP binario (BILP), donde las x_j son las variables de decisión asociadas a cada caso de prueba y, para el primer escenario, $c_{ij} = 1/m$:

$$\text{mín} \left\{ \sum_{1 \le j \le n} \left(x_j \sum_{1 \le i \le m} c_{ij} \right) \middle| \forall i \in [1, m] \sum_{1 \le j \le n} e_{ij} \ge 1 \right\} \quad (1)$$

Para ello se ha programado un algoritmo en C++ que realiza la reducción al BILP correspondiente, lo resuelve mediante CPLEX, comprueba si este declara la solución producida como óptima e informa en caso contrario.

4. Resultados experimentales

Para los experimentos se dispone de cinco composiciones[2] WS-BPEL que mantiene el grupo UCASE. Para todas ellas se calcula la solución S con el mínimo número de casos de prueba y la cobertura de mutación máxima. Los tamaños y tiempos[3] son los siguientes:

	A1	A2	A3	A4	A5
$\|M\|$	96	890	213	508	3647
$\|T\|$	8	34	19	37	95
$\|S\|$	3	17	12	8	64
Tiempo (s)	0,01	0,03	0,02	0,02	2,80

El problema más complejo (A5) se ha resuelto también empleando como métrica el coste de ejecución. Se ha obtenido una solución óptima en tan solo 2,82 s que permite mantener la máxima cobertura de mutación y minimiza el tiempo total de prueba, que se reduce de 185,80 h a 117,32 h.

[2] https://neptuno.uca.es/redmine/projects/wsbpel-comp-repo/repository.
[3] Tiempos medidos en un Intel Core i5 5200U con 2,20 GHz y 8 GiB DDR3L

5. Conclusiones y trabajo futuro

La prueba de composiciones WS-BPEL puede presentar un elevado coste de ejecución por diversos factores: la necesidad de ejecutar cada caso de prueba en múltiples escenarios, la complejidad de algunos servicios o la propia naturaleza de la ejecución de las composiciones. La ejecución implica el despliegue y ejecución de una serie de servicios, servidores web y de aplicaciones, que han de realizarse para cada caso de prueba, lo que hace que al coste intrínseco a la ejecución de la prueba haya que sumar un coste fijo nada despreciable.

Aunque este trabajo se centra en las aplicaciones a la prueba de composiciones de servicios con WS-BPEL, la técnica propuesta es lo suficientemente general como para poder utilizarse con cualquier otro lenguaje de programación o marco de prueba, siempre que la ejecución de cada caso de prueba permita distinguir qué características del SUT se ejercitan. El rendimiento es bueno incluso cuando para la composición más compleja disponible en este estudio (A5) se mantiene la máxima cobertura de mutación y se minimiza no solo el número de casos de prueba, sino también el tiempo total de prueba. En el futuro se ampliarán los experimentos a un mayor número de composiciones.

Referencias

1. Arito, F., Chicano, F., Alba, E.: On the application of SAT solvers to the test suite minimization problem. LNCS, vol. 7515, pp. 45–59. Springer (2012)
2. Black, J., Melachrinoudis, E., Kaeli, D.: Bi-criteria models for all-uses test suite reduction. In: Proc. 26th Int. Conf. on Soft. Engineering. pp. 106–115 (2004)
3. Estero-Botaro, A., Palomo-Lozano, F., Medina-Bulo, I., Domínguez-Jiménez, J.J., García-Domínguez, A.: Quality metrics for mutation testing with applications to WS-BPEL compositions. Softw. Test., Verif. Reliab. 25(5–7), 536–571 (2015)
4. Fischer, K.: A test case selection method for the validation of software maintenance modifications. In: Proceedings of International Computer Software and Applications Conference. pp. 421–426. IEEE Computer Society Press (1977)
5. Li, D., Jin, Y., Sahin, C., Clause, J., Halfond, W.G.J.: Integrated energy-directed test suite optimization. In: ISSTA 2014: Proceedings of the 2014 International Symposium on Software Testing and Analysis. pp. 339–350. ACM (2014)
6. Lopez-Herrejon, R.E., Chicano, F., Ferrer, J., Egyed, A., Alba, E.: Multi-objective optimal test suite computation for software product line pairwise testing. In: 29th IEEE Int. Conf. on Software Maintenance. pp. 404–407. IEEE (2013)
7. OASIS: Web Services Business Process Execution Language 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (2007)
8. Wang, L., Wan, R., Wang, M., Li, M.: Generating small combinatorial test suite via lp. In: WISA'09: Proc. of the 2009 Int. Symp. on Web Information Systems and Applications. pp. 226–229 (2009)
9. Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: a survey. Software Testing, Verification and Reliability 22(2), 67–120 (2012)
10. Zhong, H., Zhang, L., Mei, H.: An experimental study of four typical test suite reduction techniques. Information and Software Technology 50(6), 534–546 (2008)

Generating optimized configurable business process models in scenarios subject to uncertainty

Andrés Jiménez-Ramírez[1], Barbara Weber[2], Irene Barba[1], Carmelo Del Valle[1]

[1] Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, Spain
{ajramirez, irenebr, carmelo}@us.es
[2] Department of Computer Science, University of Innsbruck, Austria
Technical University of Denmark, Denmark
bweb@dtu.dk

Resumen. *Contexto:* La calidad de los modelos de procesos de negocio (i.e., artefactos software que capturan las relaciones entre las unidades organizacionales de un negocio) es esencial para la mejora en la gestión de los procesos de negocio. Sin embargo, dicha modelización es llevada a cabo, normalmente, a mano. Esto supone ya un desafío y consume mucho tiempo cuando (1) existe incertidumbre en la entrada, (2) las actividades están relacionadas, y (3) la asignación de recursos es necesaria. Cuando se incluyen, además, requisitos de optimización teniendo en cuenta la flexibilidad y la robustez, la tarea pasa a ser aún más complicada, pudiendo dar lugar a modelos no optimizados, errores y falta de flexibilidad. *Objetivo:* Para facilitar el trabajo manual y para mejorar los modelos resultantes en escenarios que están sujetos a incertidumbre, proponemos una solución soportada por una herramienta para generar de manera automática modelos configurables de procesos de negocio a partir de especificaciones declarativas considerando todos los requisitos anteriores. *Método:* Primero, el escenario es modelado a través de un lenguaje declarativo que permita al analista especificar la variabilidad e incertidumbre del escenario. A continuación, un conjunto de planes de ejecución optimizados –cada uno representando una potencial ejecución alternativa— se generan a partir del anterior modelo considerando la incertidumbre de entrada. Finalmente, para hacer frente a esa incertidumbre en tiempo de ejecución, un modelo configurable y flexible de proceso de negocio se crea a partir de dichos planes optimizados. *Resultados:* Para validar la propuesta, hemos realizado un caso de estudio basado en un escenario real que está sujeto a incertidumbre. Los resultados indican que nuestra contribución mejora el rendimiento real del negocio y que los modelos generados suportan la mayor parte de la incertidumbre inherente al negocio. *Conclusiones:* Este enfoque selecciona automáticamente la mejor parte de la variabilidad de una especificación declarativa. Al contrario que en otros trabajos, nuestra propuesta considera la incertidumbre de la entrada, la optimización de múltiple funciones objetivo, y las perspectivas de control del flujo y de recursos. No obstante, este trabajo presenta algunas limitaciones: (1) está centrado en la perspectiva de control del flujo pero no se ha abordado por completo la perspectiva de datos y (2) las propiedades del modelo necesitan ser estimadas.

An approach for the evolutionary discovery of software architectures

Aurora Ramírez, José Raúl Romero y Sebastián Ventura

Dpto. de Informática y Análisis Numérico
Universidad de Córdoba, Campus de Rabanales, 14071 Córdoba, España
{aramirez, jrromero, sventura}@uco.es

Resumen Las arquitecturas software representan artefactos de análisis de gran importancia en el contexto de los proyectos software, ya que reflejan los principales bloques funcionales del sistema. A su vez, ofrecen artefactos de análisis de alto nivel de abstracción que resultan útiles cuando los arquitectos necesitan analizar la estructura de un sistema en funcionamiento. Habitualmente, los arquitectos realizan este proceso de forma manual, guiados por sus propias experiencias previas. Aún así, esta tarea puede resultar muy tediosa cuando el diseño real queda oculto tras modificaciones continuas e incontroladas. Desde la reciente aparición de la ingeniería del software basada en búsqueda (*search based software engineering*), se han formulado en ingeniería del software multitud de tareas como problemas complejos de búsqueda y optimización, encontrando así la computación evolutiva una nueva área de aplicación. Este trabajo explora el diseño de un algoritmo evolutivo para el descubrimiento automático de la arquitectura subyacente a un sistema software. Se ha dedicado un importante esfuerzo a la creación de un procedimiento genérico plenamente orientado al arquitecto software. En este sentido, la selección de una codificación comprensible, una función de *fitness* inspirada en medidas precisas propias del diseño software, así como un operador genético que simula transformaciones arquitectónicas constituyen las características más importantes del enfoque de búsqueda propuesto. Finalmente, se ha realizado un completo estudio de parámetros y una experimentación sobre aplicaciones software reales con el objetivo de obtener un modelo evolutivo genérico que ayude a los ingenieros software durante el proceso de toma de decisiones.

Self-adaptation of mobile systems driven by the Common Variability Language

Gustavo G. Pascual, Mónica Pinto, and Lidia Fuentes

CAOSD Group, Universidad de Málaga, Andalucía Tech, Spain
{horcas, pinto, lff}@lcc.uma.es

Abstract. El contexto de ejecución de los sistemas pervasivos o de computación móvil cambia continuamente, y las aplicaciones deben poder adaptarse a estos cambios. Para ello, es necesario proporcionar el soporte adecuado para que las aplicaciones reaccionen a estos cambios durante su ejecución. La mayoría de las propuestas existentes para el desarrollo de sistemas auto-adaptables implementan un servicio de reconfiguración que recibe como entrada la lista de todas las configuraciones posibles y los planes para cambiar en tiempo de ejecución de una configuración a otra. En este artículo presentamos una alternativa a estas propuestas que genera de forma automática, y en tiempo de ejecución, tanto las distintas configuraciones válidas de una aplicación como los planes de reconfiguración. Con nuestra propuesta las configuraciones generadas se optimizan con respecto a distintos criterios, como son la funcionalidad y el consumo de recursos (p.ej. batería o memoria). Esto se consigue: (1) modelando la variabilidad arquitectónica en tiempo de diseño, utilizando para ello el lenguaje CVL (Common Variability Language), y (2) usando un algoritmo genético que encuentra configuraciones cuasi-óptimas en tiempo de ejecución, usando para ello la información proporcionada por el modelo de variabilidad. La propuesta se presenta y se evalúa a través de un caso de estudio, mostrando que es eficiente y apropiada para dispositivos con recursos limitados.

Ingeniería Web y Sistemas Pervasivos

Ingeniería Web y Sistemas Pervasivos
Coordinadores: Elena Navarro y Roberto Rodríguez Echeverría

Antonio Arias, Jesús M. Hermida y Santiago Meliá. *Desarrollando una fachada de servicios REST/SOA para aplicaciones SOFEA aplicando una aproximación MDE.* (Corto)

Miriam Gil, Vicente Pelechano, Joan Fons y Manoli Albert. *Involucrando al humano en el bucle de control de sistemas auto-adaptativos*. (Corto)

Roberto Rodríguez-Echeverría, Eneas Macías y José María Conejero: *Una herramienta de programación para usuarios finales de aplicaciones móviles basadas en datos abiertos.* (Demo)

Oscar Díaz y Cristóbal Arellano: The Augmented Web: Rationales, *Opportunities, and Challenges on Browser-Side Transcoding*, ACM Transactions on the Web 9(2), Article No. 8, May 2015. (Relevante)

Patricia Pons, Alejandro Catala, Javier Jaen: *Customizing Smart Environments: A Tabletop Approach*. Journal of Ambient Intelligence and Smart Environments, 7(4): 511-533, July 2015. (DOI: http://doi.org/10.3233/AIS-150328). (Relevante)

Javier Miranda, Niko Mäkitalo, Jose Garcia-Alonso, Javier Berrocal, Tommi Mikkonen, Carlos Canal y Juan M. Murillo: *From the Internet of Things to the Internet of People*, IEEE Internet Computing, 19(2), March-April, 2015, 40-47 (DOI: 10.1109/MIC.2015.24) (Relevante)

J. M. Gascueña, Elena Navarro, Patricia Fernández-Sotos, Antonio Fernández-Caballero, Juan Pavón: *IDK and ICARO to develop multi-agent systems in support of Ambient Intelligence*, Journal of Intelligent & Fuzzy Systems (DPI:10.3233/IFS-141200 28(1): 3-15). (Relevante)

José Miguel Morales, Elena Navarro, Pedro Sánchez, Diego Alonso: *Un experimento controlado para evaluar la entendibilidad de KAOS e i* para el modelado de sistemas Teleo-Reactivos*, Journal of Systems and Software 100: 1-14 (2015) (DOI:10.1016/j.jss.2014.10.010). (Relevante)

Desarrollo de una fachada de servicios REST/SOA para aplicaciones SOFEA mediante una aproximación MDE

Antonio Arias[1], Jesús Hermida[2] y Santiago Meliá[1]

[1]Universidad de Alicante, Spain
{aarias, santi}@dlsi.ua.es
[2]European Commission Joint Research Centre, Ispra, Italy
jesus.hermida@jrc.ec.europa.eu

Resumen. En los últimos años, en el desarrollo Web se ha producido un cambio drástico a nivel arquitectónico con la aparición de las arquitecturas de presentación basadas en servicios (SOFEA), motivado en gran medida por la aparición de los dispositivos móviles. Este estilo arquitectónico propone el uso de fachadas de servicios con un número reducido de operaciones y que ensamblan datos complejos para reducir el número de invocaciones. En este sentido, requieren de clases ensambladoras que transformen las entidades de negocio en los objetos ensamblados, dificultando así la mantenibilidad del código de la fachada.

Para resolver este problema, este trabajo presenta un modelo de servicios basado en la propuesta OOH4RIA. Este modelo, mediante una notación textual, permite modelar una fachada REST o SOA publicando un conjunto de operaciones del modelo de dominio. Además, introduce aspectos de seguridad y una estructuración de los datos basada en el patrón Transfer Object Assembler.

1 Introducción

En los últimos años la Web se ha producido un cambio drástico a nivel arquitectónico con la aparición de las llamadas arquitecturas SOFEA (Service Oriented Front End Architecture). En ellas, se exige una separación clara entre la parte cliente, ejecutada completamente en el navegador mediante código HTML/JavaScript/CSS, y una parte servidora sin estado que normalmente sirve las peticiones del cliente mediante una fachada REST distribuida.

Dicha fachada posee una doble función: la de ofertar un subconjunto de las operaciones de la lógica de negocio a un cliente final, y la de establecer qué datos serán intercambiados entre el cliente y el servidor. Las arquitecturas SOFEA necesitan reducir al mínimo el número de invocaciones distribuidas al servidor debido a su elevado coste temporal. En este sentido, promulgan una fachada que permita independizar a la interfaz de usuario de cómo los datos están estructurados en la parte servidora, y que, para ello, envíe los datos que se desean mostrar de forma ensamblada. El patrón de diseño Transfer Object Assembler [1] es ampliamente utilizado para resolver dicho aspecto.

adfa, p. 1, 2011.

Este patrón permite realizar un montaje de una estructura de datos compleja que es enviada a la interfaz de usuario. De esta forma, se reduce el código en el cliente al enlazarse directamente los datos enviados desde la parte servidora con los controles de usuario. Sin embargo su principal problema es la mantenibilidad, ya que requiere de un gran número de clases ensambladoras que realicen la conversión entre las entidades de negocio y los objetos ensamblados.

A pesar que se han definido otras propuestas basadas en MDE para obtener fachadas REST [5, 6], se centran normalmente en representar las operaciones y dejan de un lado la estructuración de los datos.

Para resolver esta carencia, este trabajo presenta un nuevo DSL (domain-specific language) textual denominado modelo de Servicios (Service Model), a partir del cual se pueden generar fachadas tanto REST como SOA. El modelo de servicios establece qué operaciones serán ofertadas y cómo se ensamblan los objetos DTO (Data Transfer Object) intercambiados entre el servidor y el cliente.

El modelo de servicios ha sido incluido en el proceso de desarrollo de OOH4RIA [3] reemplazando al modelo de Navegación. Para ello, se ha definido una nueva notación textual, metamodelo y transformaciones modelo a texto que generen las fachadas REST usando la tecnología WebAPI de .NET o SOA con WCF de .NET. Por falta de espacio, en este trabajo nos centramos en la fachada REST.

2 Modelo de Servicios

El modelo de servicios es un DSL independiente de plataforma (PIM) con una notación textual que permite especificar fachadas de servicios tanto REST como SOA. Este modelo define qué subconjunto de las operaciones y datos de la lógica de negocio orientada a objetos que serán accesibles por un conjunto de usuarios (o roles). El modelo de servicios tiene como elemento central el Service Class (clase de servicio). Una Service Class define a) una estructura básica de intercambio de datos entre cliente y servidor y b) las operaciones que se permiten sobre los datos. Es posible enlazar diferentes Service Class de modo que se puedan crear estructuras de intercambio más complejas. Cada Service Class ofrece una vista de los datos y operaciones de una clase de dominio de OOH4RIA.

2.1 El Modelo de Servicios en acción

En la Figura 1 se presenta un extracto del modelo de servicios para una red social. El primer elemento que se define es el propio ServiceModel, llamado APIRedSocial, que contiene al resto de elementos. En este ejemplo, el ServiceModel está asociado con el modelo de dominio de OOH4RIA (redsocial) que define una lógica de negocio orientada a objetos. A continuación se define el ServiceClass UsuarioRegistrado que

apunta a la clase Usuario. Es importante recordar, que se puede definir una o más service class sobre la misma clase de dominio, lo que permite establecer diferentes perfiles de uso según el tipo de usuario o el servicio en cuestión.

```
1  ServiceModel APIRedSocial [redsocial]
2  {
3    @secure (Role: [Admin, Usuario], User:*)
4    ServiceClass UsuarioRegistrado [redsocial.Usuario]
5    {
6      attributes
7     {
8       nombreUsuario [nombre];
9       mensajesEnviados  [-> asocMenEnv.mensajesEnviados] : Mensaje;
10      mensajesRecibidos [-> asocMenRec.mensajesRecibidos] : Mensaje;
11    }
12    operations
13    {
14      nuevosAmigos[relationer (p_usuario_oid:Integer, p_amigos_oid:List<Integer>)]:void
15      dameAmigos [-> asocAmigo.amigos (pag:10)] : Amigo;
16    }
17  }}
```

Figura 1. Fragmento del modelo de servicios de Smartloto

En la siguiente dirección se puede descargar el ejemplo completo: http://mde.dlsi.ua.es/RedSocial/APIRedSocial.tnm

2.2 Propiedades del Servicio

Como se puede apreciar en la figura 1, las clases de servicio se estructuran en dos secciones principales: attributes y operations. El modelo de servicios permite no solo filtrar los datos enviados por la fachada de servicio sino estructurarlos para formar objetos complejos utilizados. Para ello, en la sección attributes, se definen un conjunto de atributos de servicio que: a) se crean a partir de atributos del dominio (*ejemplo: atributo nombreUsuario*), o b) enlazan una clase de servicio con otra para crear objetos de datos más comp*lejos (ejemplo: mensajesEnviados y mensajesRecibidos*). En el segundo caso, los datos de la clase UsuarioRegistrado se enviarán junto a las dos listas de mensajes evitando dos llamadas más al servidor.

Basándose en la distinción que hace GRASP [2], en la sección operations se pueden publicar operaciones con responsabilidades de dos tipos. Por un lado, se publican las responsabilidades de hacer, basadas en las operaciones de la capa de negocio (en este caso la clase de domino, *ejemplo: operación nuevoAmigo*). Por otro lado, el modelo permite definir las responsabilidades de conocer, derivadas de la posibilidad de obtener información de los objetos de dominio y sus relaciones (*p.ej. operación dameA-*

migos). El modelo de servicios permite entre otros especificar si el resultado de la operación es paginado o si un valor de retorno es un ServiceClass, facilitando así el filtrado de los datos devueltos.

El modelo de Servicios permite definir qué restricciones de acceso a datos y operaciones se aplicarán por cada ServiceClass. Con la directiva *@secure* antes de la definición de la clase, es posible restringir el perfil de los usuarios (o incluso el usuario en particular) que podrán invocar los servicios o acceder a los datos (incluso desde otra clase de servicio).

3 Generación de Servicios REST

Para darle soporte a la propuesta, se han desarrollado un conjunto de transformaciones modelo a texto que partiendo del modelo de servicios que permite generar una fachada REST usando WebAPI de .NET. La fachada REST de la aplicación RedSocial del ejemplo se publicada en Azure en la siguiente dirección: http://redsocialrest.azurewebsites.net/swagger

Por cada ServiceClass, las transformaciones diseñadas generan: 1) un conjunto de operaciones de la fachada (siguiendo el estándar HTTP en cuanto a los métodos GET/POST/PUT/DELETE usados y los códigos de estado devueltos); 2) un DTOA (siguiendo el patrón Transfer Object Assembler), que será el parámetro o el valor de retorno de las operaciones generadas; y 3) una clase ensambladora (siguiendo el patrón Transfer Object Assembler), que creará el DTOA a partir de los datos manejados en la lógica de negocio. El editor del modelo de servicios y las transformaciones diseñadas fueron implementados en OIDE [4].

Referencias

1. Alur, D., Malks, D., Crupi, J., Booch, G., & Fowler, M. *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*. Sun Microsystems, Inc. 2003
2. Larman, C. (2005). *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*. Pearson Education India. 2005.
3. Meliá, S., Gomez, J., Perez, S., & Diaz, O. A model-driven development for GWT-based Rich Internet Applications with OOH4RIA. *ICWE 2008*. (pp. 13-23).
4. Meliá, S., Martínez, J. J., Mira, S., Osuna, J. A., & Gómez, J. *An eclipse plug-in for model-driven development of rich internet applications* (pp. 514-517). ICWE 2010.
5. Pautasso, C., Ivanchikj, A., & Schreier, S.. Modeling RESTful Conversations with Extended BPMN Choreography Diagrams. In Software Architecture (pp. 87-94). (2015)
6. Valverde, F., & Pastor, O. (2009). Dealing with REST services in model-driven web engineering methods. *V Jornadas Científico-Técnicas en Servicios Web y SOA, JSWEB*, 243-250.

Involucrando al humano en el bucle de control de sistemas auto-adaptativos

Miriam Gil, Vicente Pelechano, Joan Fons, and Manoli Albert

Centro de Investigación PROS, Universitat Politècnica de València,
Camí de Vera s/n, 46022 Valencia, Spain
{mgil,pele,jjfons,malbert}@pros.upv.es

Abstract. La auto-adaptación juega un papel clave en los sistemas software del futuro, los cuales están formados por complejos ecosistemas heterogéneos teniendo que ser capaces de adaptarse continua y autónomamente, en tiempo de ejecución, a su contexto (nuevas condiciones del entorno, situaciones impredecibles, necesidades cambiantes de los usuarios, nuevos recursos, etc.). Aunque estas adaptaciones deben gestionarse de forma autónoma, la experiencia demuestra que los humanos no pueden excluirse completamente del bucle de adaptación, ya sea para solucionar conflictos difíciles de resolver autónomamente o para mejorar las estrategias de adaptación con su realimentación. En este artículo se abre una línea de trabajo para involucrar al humano en el bucle de control de los sistemas auto-adaptativos ("human in the loop") de forma que éstos puedan participar en la toma de decisiones de adaptación, siempre intentando maximizar la autonomía y evitar sistemas intrusivos y molestos.

Keywords: Auto-adaptación, Computación Autónoma, "Human in the Loop", Interacción Persona-Ordenador, Experiencia de Usuario

1 Introducción

El "mundo inteligente" del futuro se está diseñando como complejos ecosistemas compuestos por una amplia variedad de dispositivos y servicios distribuidos, de naturaleza claramente móvil y ubicua, y en constante evolución tecnológica. Ante esta situación, parece clara la necesidad de desarrollar sistemas que sean capaces de adaptarse autónomamente en tiempo de ejecución a: nuevas condiciones del entorno en el que operan, situaciones impredecibles, necesidades cambiantes de los usuarios, nuevos tipos de dispositivos, interfaces de usuario o tecnologías con las que interactuar o nuevos servicios que consumir. La auto-adaptación surge como una solución para la gestión de este tipo de sistemas software que permitan hacer realidad las Ciudades Inteligentes, y los Coches Autónomos, entre otros [2].

Aunque las soluciones completamente autónomas han resultado satisfactorias en muchos dominios de aplicación, la capacidad de estos sistemas para proporcionar servicios de confianza en presencia de cambios inesperados, ya sea en su entorno (p. ej. la disponibilidad de recursos, problemas en la monitorización del contexto) o en el propio sistema (p. ej. aparición de fallos, conflictos en la satisfacción de objetivos), se ve afectada cada vez más por su creciente complejidad, así como por

la naturaleza dinámica e impredecible de los entornos en los que tienen que operar. Esto puede provocar que el sistema auto-adaptativo (SAS) sea incapaz de gestionar o dar respuesta satisfactoria a ciertas adaptaciones. Como mecanismo para dar una solución a este problema, parece interesante hacer partícipe al humano para que ayude al sistema ante situaciones de conflicto difíciles de resolver autónomamente y mejorar las estrategias de adaptación con su feedback ("human in the loop") [4][3]. Por ejemplo, por ahora los humanos son mejores que los ordenadores en reconocer caras en multitudes o conducir en situaciones complejas o de emergencia.

Normalmente, los sistemas auto-adaptativos se basan en el uso de bucles de control, los cuales son los encargados de monitorizar, analizar, decidir y actuar sobre un sistema base para gestionar de manera continua los eventos que le ocurren al propio sistema o a su entorno, y adecuarse a estas situaciones por sí mismo [2]. En este sentido, los sistemas pueden beneficiarse de la intervención humana (la cual aporta criterio o información para poder decidir) mediante [5]: (1) el rol de *sensor* por parte del humano, aportando información imprecisa, difícil de monitorizar o de deducir (p. ej. indicando si ha ocurrido una situación anómala o ajustando algún valor), (2) la consulta al humano en el proceso de *toma de decisiones* (p. ej. para resolver un conflicto de objetivos) o (3) utilizando a los humanos como *efectores* del sistema para ejecutar adaptaciones (p. ej. cuando la adaptación implica cambios físicos que no se pueden automatizar). Aún cuando los sistemas no requieran de la intervención humana, es muy importante que éstos muestren su estado y su comportamiento a los usuarios para proporcionar *transparencia*, *comprensibilidad* y *confianza*, y evitar la apariencia de "caja negra mágica" [9].

El presente trabajo en curso pretende dar una solución, desde el ámbito de la Ingeniería del Software, que permita afrontar el reto de involucrar al humano en el bucle de control de los SAS para que participe en la toma de decisiones de adaptación cuando sea necesario, intentando interrumpirlo lo mínimo para evitar SAS intrusivos y/o molestos. Además, para involucrar adecuadamente al humano y conseguir una buena experiencia de usuario es necesario abordar un tercer aspecto muy importante: la interacción del humano con el SAS.

2 Los Humanos en los Sistemas Auto-adaptativos: Limitaciones Existentes

Aunque conseguir un alto grado de autonomía es una condición necesaria para el crecimiento de los SAS, es importante establecer un equilibrio entre el grado de autonomía y la intervención humana para evitar llegar a resultados no deseados o una mala experiencia de usuario [8]. Como consecuencia, la intervención humana en los SAS puede ser discutida desde dos puntos de vista: 1) **el grado de automatización** de los mecanismos de adaptación (relacionado con el nivel de autonomía conseguido), y 2) **la calidad de interacción** del sistema con los humanos.

Por una parte, desde el punto de vista del grado de autonomia, Barkhuss y Dey [1] examinaron el grado de autonomía que las aplicaciones adaptativas debían tener desde el punto de vista de los usuarios. Definieron tres niveles de interactividad: *personalización* (permiten al usuario especificar el comportamiento de la aplicación

dada una situación), *conciencia del contexto pasiva* (presentan el contexto actualizado al usuario y le permiten a éste decidir cómo cambiar el comportamiento), y *conciencia del contexto activa* (el sistema cambia de forma autónoma el comportamiento en base a la información de contexto). De este estudio observaron que los usuarios preferían las características adaptativas a la personalización, pero al mismo tiempo con los SAS experimentaron una falta de control. Por tanto señalaron la necesidad de **gestionar un equilibrio entre el control del usuario y la autonomía del sistema** en tiempo de ejecución.

Por otra parte, para incrementar la experiencia de usuario y conseguir la interacción óptima deseada, Russell et al. [8] apuntaron la necesidad de un cambio en el diseño de sistemas autónomos pensando desde el principio en la interacción con los humanos, y soportando: (a) modelos de **gestión de la atención del usuario** para evitar interrumpir y abrumar a los usuarios, (b) **transparencia y control** del sistema para el post-análisis de fallos, la comprensión, y su recuperación y (c) **mecanismos de "feedback" del comportamiento** del sistema que permitan a los usuarios entender lo que el sistema está haciendo con cierta medida de confianza.

3 Objetivos de la Propuesta

Para abordar las limitaciones existentes e involucrar al usuario en los SAS mediante la interacción apropiada, proponemos diseñar una solución que persiga los siguientes objetivos:

1. **Gestionar la atención del usuario y la molestia.** Una forma de gestionar la atención del usuario y desarrollar sistemas menos molestos es utilizando interacciones implícitas, aquellas que ocurren sin que el usuario sea consciente. Para ello, nos basaremos en un framework para diseñar interacciones implícitas [6] que divide el espacio de interacciones en dos dimensiones (ver ejes y punto 1 en Figura 1): la *iniciativa* (quién inicia la interacción, el usuario o el sistema) y la *atención* (grado de atención requerido).
2. **Soportar diferentes tipos de participación y una autonomia ajustable.** Los sistemas deben proporcionar mecanismos para ajustar dinámicamente el grado de control del usuario (tipo de participación) dependiendo de diferentes circunstancias (p. ej., tareas críticas, comportamiento inesperado, incertidumbre en la monitorización, etc.). En nuestra propuesta, el framework define varios tipos de participación (se podrían definir tantos niveles como se necesitase) representados por los cuadrantes (ver Fig. 1). Para soportar un ajuste de la autonomía en tiempo de ejecución definiremos transiciones entre los tipos de participación que adapten la participación en función del conflicto a resolver.
3. **Soportar la transparencia, control, y feedback.** Para soportar estas características y mantener la confianza de los usuarios es esencial proporcionarle explicaciones de lo que está pasando en el sistema [4]. Para ello, utilizaremos el modelo de explicaciones que se presenta en [7] el cual genera explicaciones a partir de los modelos de conocimiento (*que ha pasado, por qué, por qué no*, etc.). Además, el control y el feedback lo soportamos mediante el framework para el diseño de interacciones implícitas (ver punto 3 en Figura 1), diseñando la participación del usuario en los cuadrantes correspondientes.

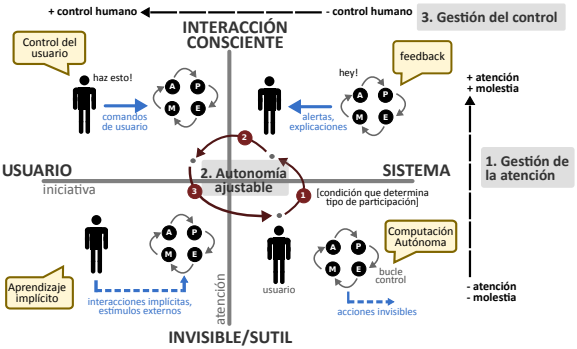


Fig. 1. Marco conceptual para el diseño de la participación del usuario.

4 Conclusiones

En este trabajo se ha presentado una primera aproximación para involucrar al usuario en el bucle de control de los SAS, identificando los objetivos perseguidos en nuestra propuesta. El principal objetivo de los autores consiste en fomentar la discusión sobre este reto de investigación, obtener retroalimentación para mejorar la propuesta y buscar posibles colaboraciones en las jornadas.

Agradecimientos. Este trabajo ha sido financiado por el Ministerio Español de Economía y Competitividad bajo el proyecto SMART-ADAPT TIN2013-42981-P.

References

1. Barkhuus, L., Dey, A.: Is context-aware computing taking control away from the user? Three levels of interactivity examined. UbiComp 2003 (2003)
2. Brun, Y., Serugendo, G., Gacek, C., Giese, H.: Engineering self-adaptive systems through feedback loops. Software engineering for self-adaptive systems (2009)
3. Cámara, J., Moreno, G., Garlan, D.: Reasoning about Human Participation in Self-Adaptive Systems. In: SEAMS 2015. pp. 146–156 (2015)
4. Cheng, B., De Lemos, R., Giese, H., Inverardi, P.: Software engineering for self-adaptive systems (2009)
5. Garlan, D.: A 10-year perspective on software engineering self-adaptive systems (keynote). SEAMS 2013 pp. 2–2 (2013)
6. Ju, W., Leifer, L.: The design of implicit interactions: Making interactive systems less obnoxious. Design Issues (2008)
7. Lim, B.Y., Dey, A.K.: Toolkit to support intelligibility in context-aware applications. In: Ubicomp '10. pp. 13–22 (2010)
8. Russell, D.M., Maglio, P.P., Dordick, R., Neti, C.: Dealing with ghosts: Managing the user experience of autonomic computing. IBM Systems Journal 42(1), 177–188 (2003)
9. Stumpf, S., Burnett, M., Pipek, V., Wong, W.K.: End-user Interactions with Intelligent and Autonomous Systems. In: Human Factors CHI '12. pp. 2755–2758 (2012)

Una herramienta de programación para usuarios finales de aplicaciones móviles basadas en datos abiertos ⋆

Roberto Rodríguez-Echeverría, Eneas Macías, and José M. Conejero

Quercus Software Engineering Group, Universidad de Extremadura,
{rre, enmaciasm, chemacm}@unex.es

Resumen. Durante los últimos años los teléfonos inteligentes han experimentado un rápido crecimiento. De manera que, actualmente, los dispositivos móviles constituyen la opción más frecuente de acceso a internet. De hecho, casi se puede decir que existe una aplicación móvil para cada aspecto de nuestra vida tanto profesional como personal. Por otro lado, últimamente, impulsado por las políticas internacionales de transparencia, se han puesto a disposición pública un gran número de recursos de datos abiertos. Datos abiertos de turismo, de meteorología y geográficos, por citar algunos, se han puesto disponibles para su reutilización por parte de instituciones públicas de todos los niveles. Sin embargo, a pesar de las acciones realizadas para la generación de aplicaciones que exploten estas nuevas fuentes de datos, la gran mayoría está enfocada en datos de un solo dominio y, normalmente, en datos de un solo conjunto de datos. Por lo tanto, los usuarios finales se ven forzados a utilizar varias aplicaciones al mismo tiempo para poder satisfacer sus necesidades particulares. En este trabajo, proponemos una herramienta de desarrollo de aplicaciones por parte de usuarios finales para la creación de aplicaciones basadas en datos abiertos personalizadas.

Palabras clave: Programación por Usuarios Finales, Aplicaciones Móviles, Datos Abiertos

1 Introducción

Durante la última década, las tecnologías de la información (TI) han supuesto una disrupción socioeconómica que ha calado en todos los ámbitos de la sociedad. Obviamente, esta disrupción TI en el mundo se debe a muchos factores, pero se pueden resaltar algunos como principales catalizadores: Internet, la Web 2.0, las redes sociales, los dispositivos móviles inteligentes (smartphones), la apertura de los datos [1,2] por parte de las administraciones públicas y algunas grandes empresas. Hasta ahora, animada por el auge de los smartphones, el acceso a la

⋆ Trabajo financiado la Consejería de Empleo, Empresa e Innovación del Gobierno de Extremadura, Ayuda GR15098; y el Fondo Europeo para el Desarrollo Regional (FEDER)

información abierta se ciñe fundamentalmente al uso de una u otra aplicación móvil que realiza una reutilización parcial de esos datos, que puede no satisfacer las necesidades del ciudadano, en muchos casos.

Para ejemplificar este problema, pensemos en un turista que quiere visitar la ciudad de Cáceres y presenta las siguientes necesidades: (1) quiere saber dónde hay aparcamientos en la ciudad y qué precio tienen; (2) quiere saber también si cerca de esos aparcamientos existe la posibilidad de coger otro medio de transporte (taxi, autobús u otros) para desplazarse hasta el centro de la ciudad; y (3) quiere saber qué franja horaria es mejor para visitar la parte antigua, en función de la temperatura y de los niveles de polen del día que quiere realizar la visita. Obviamente, no existe actualmente ninguna aplicación que le permita obtener la respuesta que busca mediante la composición de todos los datos disponibles, sino que tendrá que hacer uso de buscadores, leer diferentes páginas Web y hacer uso de diferentes aplicaciones móviles, para al final tener que realizar el esfuerzo de la composición de esos datos por sí mismo y, en ese momento, sin más ayuda de toda la tecnología que le rodea. Por lo tanto, aunque, como se ha comentado anteriormente, tenemos toda la tecnología necesaria (smartphone, redes 3/4G, opendata), nos falta una componente fundamental en la ecuación: la tecnología que permita al ciudadano especificar fácilmente a la máquina qué es lo que quiere hacer y cuáles son sus necesidades (los usuarios son los nuevos programadores) [3,4].

En este trabajo presentamos una propuesta de programación de aplicaciones móviles basadas en datos abiertos por parte de usuarios finales. Una de las características más innovadoras de esta propuesta es que, al contrario de otras soluciones, permite al usuario crear su aplicación directamente en su dispositivo móvil.

2 Programación móvil por usuarios finales

El desarrollo software por usuarios finales (End-User Development, EUD) es "un conjunto de métodos, técnicas y herramientas que permite a los usuarios de sistemas software, que actúan como desarrolladores software no profesionales, en algún momento crear, modificar o extender un artefacto software" [4]. El principal valor de esta aproximación es el hecho de que los usuarios finales conocen mejor que nadie su propio contexto y pueden responder en tiempo real a cambios que se puedan dar en sus respectivos dominios.

EUD es, sin embargo, inherentemente diferente del desarrollo software tradicional. La programación por usuarios finales (End-User Programming, EUP) es la fase de EUD que se centra en la creación de aplicaciones. Los usuarios finales pueden llevar a cabo esta tarea mediante diferentes estilos de interacción [5]: entornos de programación visual, programación por demostración (o ejemplo), programación por especificación y lenguajes de programación textuales, entre otros.

En este trabajo se propone seguir el estilo de programación por especificación que, básicamente, consiste en un medio de descripción de la aplicación, que

puede ser desde un lenguaje específico del dominio hasta un formulario de configuración, y de una herramienta que permita generar la aplicación final. Para satisfacer el escenario de uso comentado en la motivación del trabajo se define como requisito fundamental de la solución la necesidad de que el usuario pueda crear su aplicación en un dispositivo móvil. Por consiguiente, tanto la descripción de la aplicación como la generación de la misma se debe realizar mediante una aplicación móvil. Éste es el propósito de la aplicación yourInstantApp (yIA).

yIA propone como medio de especificación de la aplicación un asistente que consta de los siguientes 5 pasos fundamentales, ilustrados en la Figura 1:

1. Seleccionar las temáticas (categorías) de datos que le interesan al usuario, que puede seleccionarlas de una lista. Adicionalmente, el usuario puede elegir filtrar los datos por localización, por ejemplo, de manera que sólo los datos disponibles para la localización elegida de las temáticas seleccionadas se devolverán al usuario.
2. Seleccionar conjuntos de datos interesantes para el usuario a partir del resultado del paso anterior.
3. Seleccionar los datos concretos (atributos) de cada conjunto que se quieren utilizar.
4. Seleccionar una forma de visualización de los datos elegidos. Dependiendo de la naturaleza de los datos disponibles, el asistente recomendará una forma de visualización u otra. Además, cada visualización proporcionará un estilo de interacción con el usuario, pudiendo extenderse mediante la especificación de acciones personalizadas.
5. Proporcionar un nombre a la aplicación para su generación y almacenamiento.

Una vez generada, la aplicación puede ser lanzada desde dentro de yIA. Por lo tanto, yIA actúa a su vez como entorno de ejecución de las aplicaciones creadas por el usuario final. Adicionalmente, yIA proporciona al usuario un repositorio local de aplicaciones, que le permite modificar una aplicación existente o compartir aplicaciones con otros usuarios, entre otras funcionalidades.

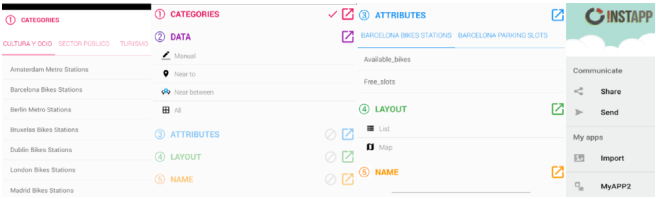


Figura 1: Especificación asistida por yIA

3 Trabajos relacionados

Como señala [6], a pesar del auge del uso de dispositivos móviles para el acceso a internet durante los últimos años, todavía no existen muchos trabajos relacionados con EUD para aplicaciones móviles. Entre los existentes podemos establecer dos dimensiones para su categorización: (1) soluciones completamente móviles o híbridas (escritorio-móvil), (2) soluciones genéricas o específicas de un dominio (p.ej. turismo). Hasta ahora gran parte de las soluciones son de carácter híbrido [7], de manera que la especificación y generación se realizan en entornos de escritorio que finalmente generar una aplicación móvil. Este tipo de soluciones, sin emgargo, no satisfacen el requisito de que la solución sea completamente móvil. Existen algunas soluciones de naturaleza móvil, como por ejemplo [8] que utiliza un lenguaje de programación textual y teclados virtuales específicos para simplificar la escritura de aplicaciones. Este trabajo propone asistir el proceso de creación de la aplicación. Además, se orienta al desarrollo de aplicaciones basadas en datos abiertos y no en ofrecer una solución totalmente genérica.

4 Conclusiones y trabajo futuro

En este trabajo se presenta un entorno de especificación y generación de aplicaciones móviles mediante el cual un usuario final puede crear en cualquier sitio y momento la aplicación que mejor satisfaga sus necesidades puntuales. Las principales líneas de trabajo actual consisten en la realización de pruebas con usuarios finales para valorar la adecuación de la solución en diferentes dominios de aplicación y la definición de lenguajes específicos del dominio para la especificación.

Referencias

1. Noor Huijboom and Tijs Van der Broek. Open data: an international comparison of strategies. *European Journal of ePractice*, pages 4–116, 2011.
2. Rob Kitchin. *The Data Revolution: Big Data, Open Data, Data Infrastructures and Their Consequences*. Sage, London, 2014.
3. Grady Booch. ICSE 2015 Keynote. `https://youtu.be/h1TGJJ-F-fE`, 2015. [Online; accessed 01-July-2016].
4. Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. End-User Development: An Emerging Paradigm. In *End User Development*, pages 1–8. Springer Netherlands, Dordrecht, 2006.
5. Bonnie A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, Cambridge, MA, USA, 1993.
6. Fabio Paternò. End User Development: Survey of an Emerging Field for Empowering People. *ISRN Software Engineering*, pages 1–11, 2013.
7. MIT. App Inventor 2. `http://appinventor.mit.edu/`, 2012. [Online; accessed 01-July-2016].
8. Balaji Athreya, Faezeh Bahmani, Alex Diede, and Chris Scaffidi. End-user programmers on the loose: A study of programming on the phone for the phone. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, pages 75–82, 2012.

The Augmented Web: Rationales, Opportunities, and Challenges on Browser-Side Transcoding

Oscar Díaz and Cristóbal Arellano

University of the Basque Country (UPV/EHU),
ONEKIN Research Group - Facultad de Informática
San Sebastián, Spain
{oscar.diaz, cristobal.arellano}@ehu.es

Resumen

Actualmente, las tecnologías de personalización Web utilizan enfoques como la categorización, la configuración o la personalización, pero no alcanzan a conseguir una experiencia plenamente individualizada. Ya que una parte creciente de nuestra actividades se realizan via Web, es de esperar que exista una mayor presión para conseguir que esta experiencia Web sea cada vez "mas individual". La transcodificación es decir, la modificación del código HTML de las páginas una vez recibidas por el navegador, abren una oportunidad para mejorar esta anhelo de "individualidad Web" a través de los plugs-in y las extensiones de navegador. Esto se ha venido en llamar "la aumentación Web" (AW). La AW es a la web lo que la realidad aumentada es al mundo físico: añadir una capa suplementaria de contenido, diseño o navegación a las páginas Web con la intención de mejorar la experiencia del usuario. Desde esta perspectiva, la AW no es tan potente como la personalización Web, ya que su alcance se limita sólo al código HTML de las páginas. Sin embargo, presenta la ventaja que esta personalización puede ser realizada por desarrolladores ajenos a los que inicialmente realizaron la página Web. Esto abre la Web a terceros que pueden "tunear" las páginas Web para sus propios fines. La existencia de miles de plugs-in que acumulan millores de descargas avalan el impacto de este enfoque. Este trabajo indaga en este fenómeno, profundizando en "el qué", "el por qué" y "el para qué" de la AW, y en los desafios pendientes. Para ello, analizamos las 45 extensiones de AW más descargadas para Mozilla Firefox y Google Chrome, haciendo una revisión sistemática de la literatura para identificar qué aspectos de la calidad recibieron mayor atención. El objetivo es sensibilizar sobre el potencial de la AW asi como señalar posible vías de investigación.

Publicado en

ACM Transactions on the Web (TWEB), Volume 9 Issue 2, pp. 1-30, 2015
http://dl.acm.org/citation.cfm?doid=2735633

Customizing Smart Environments: A Tabletop Approach*

Patricia Pons, Alejandro Catala, y Javier Jaen

Grupo ISSI, Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, Valencia, Spain
{ppons@dsic.upv.es, acatala@dsic.upv.es, fjaen@upv.es}

Abstract. La especificación de comportamiento en entornos pervasivos puede resultar compleja para usuarios no expertos. Este artículo presenta un editor de reglas puramente visual basado en expresiones de flujos de datos para superficies interactivas, y describe el experimento conducido para evaluar la usabilidad de este editor en la especificación de comportamiento en el contexto de un hogar inteligente. El estudio compara la comprensión del modelo de reglas, la interfaz visual y los mecanismos de interacción de la herramienta de edición en dos grupos de participantes: programadores y no programadores, revelando que los usuarios sin conocimientos previos de programación fueron capaces de manipular sin problemas el editor de reglas propuesto.

Keywords: Reglas, definición de comportamiento, sistemas basados en eventos, personalización, entorno inteligente, superficie interactiva, lenguaje visual

* Pons, P., Catala, A., Jaen, J. Customizing smart environments: A tabletop approach. Journal of Ambient Intelligence and Smart Environments, 7(4): 511-533, July 2015. DOI: http://doi.org/10.3233/AIS-150328. JCR 2014, IF: 1.063, Computer Science – Information Technology Q2 [65/139]

From the Internet of Things to the Internet of People

Javier Miranda[1], Niko Mäkitalo[2], Jose Garcia-Alonso[3], Javier Berrocal[3], Tommi Mikkonen[2], Carlos Canal[4], and Juan M. Murillo[3]

[1] Gloin
jmiranda@gloin.es
[2] Tampere University of Technology
{niko.makitalo,tommi.mikkonen}@tut.fi
[3] Universidad de Extremadura
{jgaralo,jberolm,juanmamu}@unex.es
[4] Universidad de Málaga
canal@lcc.uma.es

Resumen El principal objetivo en el desarrollo de aplicaciones para el Internet de las Cosas consiste en integrar la tecnología en la vida diaria. Sin embargo, la forma en que esta integración se implementa actualmente tiene mucho margen de mejora. Generalmente los usuarios tienen que configurar las aplicaciones utilizando un conjunto de parámetros, y cuando el contexto de una persona cambia, esta debe reconfigurar dichos parámetros manualmente. Lo que pretendía ser una mejora de la vida diaria se convierte en ruido adicional cuando el usuario se encuentra en situaciones imprevistas. En este trabajo los autores describen una arquitectura de referencia que utiliza dispositivos móviles para mejorar la integración del Internet de las Cosas, dando paso a nuevos escenarios que permiten la evolución del Internet de las Cosas hacia el Internet de las Personas.

XXI Jornadas de Ingeniería del Software y Bases de Datos, pp. 531

IDK and ICARO to Develop Multi-agent Systems in Support of Ambient Intelligence

José M. Gascueña[1], Elena Navarro[1,2], Patricia Fernández-Sotos[3], Antonio Fernández-Caballero[1,2], Juan Pavón[4]

[1] Instituto de Investigación en Informática de Albacete (I3A), Albacete, España
[2] Universidad de Castilla-La Mancha, Departamento de Sistema Informáticos, Albacete, España
[3] Universidad de Castilla-La Mancha, Facultad de Medicina, Albacete, España
[4] Universidad Complutense de Madrid, Facultad de Informática, Departamento de Ingeniería del Software e Inteligencia Artificial, Madrid, España
Jmanuel.Gascuena@gmail.com, Elena.Navarro@uclm.es, patry333@hotmail.com, Antonio.Fdez@uclm.es, jpavon@fdi.ucm.es

Resumen. Una cuestión importante en la Inteligencia Ambiental (AmI) es el despliegue masivo de agentes inteligentes incorporados en el entorno que se adapten a los perfiles, preferencias y necesidades de los usuarios. Esta es la razón por la que el uso de metodologías y marcos de trabajo orientados a agentes es casi obligatorio para desarrollar fácilmente el software para escenarios AmI. ICARO es un marco de trabajo software que promueve el uso de diferentes patrones de organización y de comportamiento para implementar sistemas multi-agente (SMA). Su amplio uso en varios proyectos demuestra un incremento sustancial en la productividad del software. También, con el fin de reducir el esfuerzo de codificación, es habitual diseñar SMA a un nivel mayor de abstracción. En este sentido, se ha realizado la generación de código desde especificaciones SMA hacia el marco de trabajo ICARO. INGENIAS Development Kit (IDK) soporta la especificación de modelos SMA, incluyendo cualquier característica necesaria para implementar SMA con ICARO, y un conjunto de facilidades para la generación de código. Este artículo describe el desarrollo de aplicaciones AmI gracias a la integración de ICARO e IDK. Se han desarrollado dos módulos IDK: un módulo de generación de código y un módulo de soporte para la actualización de código

Palabras clave. Sistemas Multi-agente, Ingeniería del Software Orientada a Agentes, Generación de código, Inteligencia Ambiental, Smart Homes, Sistemas de Inteligencia Ambiental

Un experimento controlado para evaluar la entendibilidad de KAOS e i* para el modelado de sistemas Teleo-Reactivos [*] [†]

José Miguel Morales[a], Elena Navarro[b], Pedro Sánchez[a], Diego Alonso[a]

[a] División de Sistemas e Ingeniería Electrónica (DSIE), Universidad Politécnica de Cartagena
{josemiguel.morales, pedro.sanchez, diego.alonso}@upct.es

[b] LoUISE Research Group, Computing Systems Department, University of Castilla-La Mancha
elena.navarro@uclm.es

Abstract. Las especificaciones Teleo-Reactivas (TR) permiten a los ingenieros definir el comportamiento de sistemas reactivos considerando objetivos y cambios en el estado del entorno. Este artículo evalúa dos notaciones distintas de Ingeniería de Requisitos dirigida por Objetivos (GORE), i* y KAOS, para determinar su entendibilidad a la hora de especificar sistemas TR. Se ha desarrollado un experimento controlado con dos grupos de estudiantes de grado. Cada grupo primero analizó un modelo de requisitos de un sistema robótico móvil, especificado usando uno de los lenguajes evaluados, entonces rellenaron un cuestionario para evaluar su entendibilidad. Posteriormente, cada grupo procedió de manera similar con el modelo del otro sistema de ejemplo y el otro lenguaje. El análisis estadístico de los datos obtenidos a través del experimento muestra que la entendibiliad de i* es mayor que la de KAOS cuando se modelan sistemas TR. Ambos lenguajes son adecuados para especificar sistemas TR aunque sus notaciones necesitan adaptarse para maximizar la entendibilidad. La notación i* supera a KAOS fundamentalmente por dos razones: los modelos i* permiten representar dependencias entre agentes, objetivos y tareas, y por otro lado, las diferencias notacionales entre tareas y objetivos en i* son más evidentes que las que existen entre objetivos y requisitos en KAOS.

[*] Los autores agradecen al Ministerio Español de Economía y Competitividad el soporte proporcionado mediante los proyectos ViSel-TR (ref. TIN2012-39279), cDrone (ref. TIN2013-45920-R) e insPIre (ref. TIN2012- 34003). Este trabajo ha sido realizado en el programa "*Research Programme for Groups of Scientific Excellence at Region of Murcia*" Fundación Séneca (Agencia para Ciencia y Tecnología de la Región de Murcia, 19895/GERM/15).

[†] Artículo completo: "*José Miguel Morales, Elena Navarro, Pedro Sánchez, Diego Alonso. A controlled experiment to evaluate the understandability of KAOS and i* for modeling Teleo-Reactive systems, Journal of Systems and Software, Volume 100, February 2015, Pages 1-14*"

Proceso Software y Metodologías

Proceso Software y Metodologías
Coordinadores: Mercedes Ruiz y Agustín Yagüe

Alejandro Calderón Sánchez y Mercedes Ruiz. *Uso de Juegos Serios para la Formación en los Procesos del Ciclo de Vida y Mejora del Software.* (Completo)

Patricio Letelier y Mª Carmen Penadés. *AgileRoadmap. Un modelo y estrategia para implantación de prácticas ágiles.* (Completo)

José Carlos Sancho Núñez, Andrés Caro Lindo, Pablo García Rodríguez y Ángel Quesada: *Categorización de Actividades de Seguridad en el Desarrollo de Software.* (Corto)

Nuria Hurtado, Mercedes Ruiz, Elena Orta, Jesús Torres: *Simulación para la Toma de Decisiones en la Gestión del Proceso de Evaluación de la Usabilidad.* Information and Software Technology, Volume 57, 509–526, 2015 (DOI: 10.1016/j.infsof.2014.06.001). (Relevante)

Alejandro Calderón, Mercedes Ruiz: *Evaluación de Juegos Serios: una Revisión Sistemática de la Literatura con Aplicación en Dirección y Gestión de Proyectos Software.* Computers & Education 87: 396-422 (2015). (Relevante)

Pedro Silva, Ana Moreno y Lawrence Peters: *Software Project Management: Learning from Our Mistakes.* IEEE Software, May/June 2015, pp. 40-43. (Relevante)

Uso de Juegos Serios para la Formación en los Procesos del Ciclo de Vida y Mejora del Software

Alejandro Calderón [1] y Mercedes Ruiz [1]

[1] Departamento de Ingeniería Informática, Universidad de Cádiz.
Avenida de la Universidad de Cádiz, 10
11519 - Puerto Real (Cádiz), España.
{alejandro.calderon, mercedes.ruiz}@uca.es

Abstract. La formación en proceso de software es un tema de gran relevancia a tener en cuenta por los profesionales de la industria de la ingeniería del software en el camino hacia el desarrollo de software de calidad con éxito. Sin embargo, los estudios realizados muestran como dicha formación es altamente teórica y en la mayoría de los casos suele impartirse de forma separada a la formación en las actividades de desarrollo del software, en un entorno donde los futuros profesionales apenas adquieren experiencia práctica en la aplicación de los procesos del ciclo de vida y mejora del software. En este artículo, analizamos el uso de los juegos serios para la formación en proceso de software, y los grupos de procesos del ciclo de vida del software definidos por el estándar ISO/IEC 12207. Además, proponemos un juego serio basado en simulación para formar en dirección y gestión de proyectos software, y lo relacionamos con los procesos del estándar ISO/IEC 12207. Por último, presentamos los resultados de la evaluación realizada por expertos (n=10) sobre el uso del juego serio propuesto para la formación en proceso de software, concluyendo que su uso durante el curso ayuda a los alumnos en la adquisición de los conocimientos del proceso de software de forma práctica.

Palabras claves: Proceso de Software, ISO/IEC 12207, Juegos Serios, Formación

1 Introducción

La ingeniería del software es un área de conocimiento dentro de las Ciencias de la Computación y las Tecnologías de la Información que proporciona un conjunto de métodos, herramientas y procedimientos para el diseño, desarrollo y mantenimiento de sistemas, productos o servicios software de calidad, dentro de las limitaciones de tiempo y coste [1]. Para alcanzar este objetivo, existen estándares y normas que dan soporte a las diferentes actividades que se realizan durante todo el ciclo de vida del software, estándar ISO/IEC 15504-5 [2], y proporcionan un conjunto de procesos de software para definir estas actividades, estándar ISO/IEC 12207 [3].

En el camino del éxito del desarrollo de software de calidad, la formación que los futuros ingenieros de software reciben en el ámbito de la ingeniería del software y de

los procesos del ciclo de vida y mejora del software es un factor de gran importancia. Sin embargo, se observa como la formación en los procesos de software suele tratarse en un marco teórico basándose principalmente en el trabajo con pequeños proyectos o ejercicios de pizarra [4], donde apenas se relaciona de forma práctica con las actividades del desarrollo de software. Esto provoca que los alumnos pierdan el interés en el aprendizaje de los conocimientos de esta importante área, disminuyendo a su vez la implicación y la motivación de los mismos en el proceso de aprendizaje.

Este enfoque teórico es insuficiente para que los futuros profesionales de la ingeniería del software adquieran la experiencia necesaria con el fin de desarrollar productos o servicios software de calidad en el ámbito profesional. Es por ello, que los profesionales de la industria y reconocidas organizaciones como ACM o IEEE ponen de manifiesto la necesidad de una formación donde los alumnos adquieran sus conocimientos en un entorno práctico, donde experimenten con escenarios cercanos a la realidad profesional durante sus estudios, y donde la formación de los procesos de software, tanto práctica como teórica se traten de forma conjunta [5].

Los estudios realizados nos han permitido identificar la necesidad actual de convertir la formación en proceso de software en un tema interesante, tanto para los alumnos como para los profesionales, debido a que entre otros factores, el éxito del proceso de software depende de la formación de las personas involucradas en el desarrollo de software [6].

Los juegos serios son herramientas con el objetivo de formar a los participantes que ayudan a estos a experimentar, aprender de sus propios errores y adquirir experiencia práctica en un entorno libre de riesgos y costos. Además, los juegos serios permiten mejorar la implicación e incrementar la motivación de los alumnos en el proceso de aprendizaje [7] [8].

En este artículo, analizamos el uso de los juegos serios para la formación en los procesos del ciclo de vida y mejora del software, y describimos las características más relevantes de ProDec [9], un juego serio basado en simulación para formar y evaluar en dirección y gestión de proyectos software, relacionándolo, a su vez, con los procesos del ciclo de vida del software de la ISO/IEC 12207.

El artículo se organiza como sigue: la sección 2 expone los grupos de procesos del ciclo de vida del software definidos por el estándar ISO/IEC 12207; la sección 3 analiza la aplicación de los juegos serios para la formación en proceso de software; la sección 4 describe las características principales de ProDec y lo relaciona con los procesos del ciclo de vida del estándar ISO/IEC 12207; la sección 5 muestra los resultados de la evaluación preliminar de la idea con expertos. Por último, la sección 6 presenta las conclusiones y las líneas de trabajo futuras.

2 ISO/IEC 12207

ISO/IEC 12207 es un estándar internacional de la Ingeniería del Software que establece un marco de trabajo común para los procesos del ciclo de vida del software [3]. El estándar contiene los procesos, las actividades y las tareas que deben ser aplicadas durante la adquisición de un producto, servicio o sistema software y durante

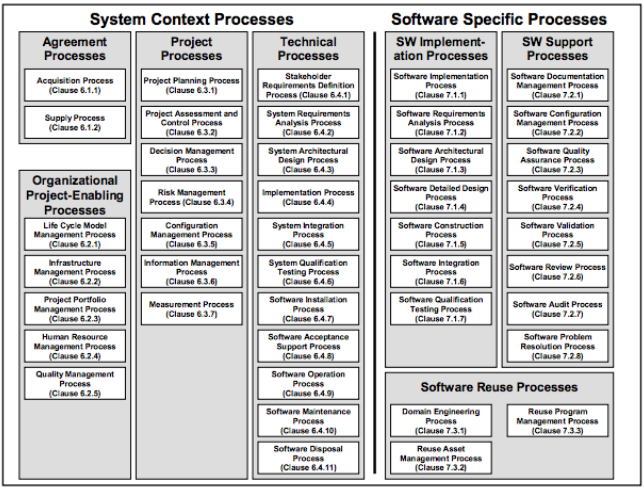


Figura 1. Procesos del ciclo de vida del software [2].

el suministro, desarrollo, operación, mantenimiento y eliminación de los productos, servicios o sistemas software. De este modo, el estándar proporciona un conjunto de procesos que abarcan todo el ciclo de vida del software desde el comienzo hasta el final del producto.

El estándar ISO/IEC 12207 define un total de 43 procesos, los cuales se dividen en siete grupos de procesos (véase Figura 1). El estándar describe el propósito de cada proceso dentro de su propio grupo y los resultados que se desean obtener con él, así como el conjunto de tareas y actividades necesarias para alcanzar dichos resultados. Estos siete grupos de procesos se clasifican a su vez en dos grupos principales referenciados por las cláusulas 6 y 7 del estándar, el cual posee una estructura compuesta por siete cláusulas y siete anexos. La cláusula 6, proporciona el contexto para tratar con un producto o servicio software de forma independiente al sistema que pertenece. Por otro lado, la cláusula 7, contiene los procesos específicos que se utilizan para el desarrollo de un producto o servicio software que forma parte de un sistema más complejo.

3 Juegos serios

Los juegos serios también llamados juegos formativos, son juegos diseñados para educar, entrenar o informar a los participantes, cuyo propósito persigue un fin más allá que la pura diversión [10]. Según Mike Zyda, podemos definir el concepto de juego serio como: "una prueba mental, de acuerdo con unas reglas específicas, que usa la diversión como modo de formación gubernamental o corporativo, con objetivos en el ámbito de la educación, sanidad, política pública y comunicación estratégica [11]."

Los juegos serios son herramientas de aprendizaje que permiten que los participantes experimenten y aprendan de sus errores. Además, permiten la adquisición de experiencia de forma segura cuando se trabaja en entornos peligrosos o de alto riesgo. El objetivo fundamental de los juegos serios es crear entornos de aprendizaje que permitan experimentar con problemas reales, en donde se pretende que el juego sirva a los participantes para experimentar y probar múltiples soluciones, explorar, descubrir la información y los nuevos conocimientos sin temor a equivocarse, pues en el juego se toman decisiones que no tienen consecuencias en el mundo real. Además, los juegos incorporan la posibilidad de ser multijugador, lo que facilita también la resolución de problemas en grupo, la colaboración, y el desarrollo de habilidades de negociación. Se aprende del juego y también de las acciones, ideas y decisiones de los demás participantes.

Por otro lado, los avances actuales en las tecnologías permiten que este tipo de juegos se desarrollen bajo múltiples plataformas de manera que proporcionan acceso y dan sentido y valor educativo al uso de los teléfonos móviles, consolas de videojuegos, los reproductores multimedia y otros dispositivos que forman parte de nuestra vida cotidiana.

Las características propias de los juegos, sus ventajas al trasladarlos al ámbito de la educación, los continuos avances en la tecnología y la necesidad de formar en el proceso de software de forma más práctica, hacen que los formadores tengan que adaptarse al uso y desarrollo de nuevas técnicas y métodos con el fin de ofrecer una enseñanza más práctica, que promueva el aprendizaje activo e interactivo y mejore la motivación en la adquisición de los conocimientos que giran en torno al proceso de software.

El análisis de la literatura realizado permite observar como a pesar de la importancia que los procesos de software tienen en el desarrollo de software de calidad, existen pocos estudios relacionados con el desarrollo de juegos serios para mejorar la educación de los futuros profesionales en el proceso de software. Entre los trabajos encontrados, destacamos los siguientes: DesignMPS [12] es un juego de ordenador diseñado para formar en el modelado del proceso de software. Problems and Programmers [4] es un juego serio basado en cartas con el fin de formar en los procesos de la ingeniería del software. Por último, Aydan, Yilmaz and O'Connor [13] proponen el desarrollo de un juego serio 3D que simule el entorno de una oficina para mejorar la aplicación de los procesos del estándar ISO/IEC 12207 en el desarrollo de software por parte de los alumnos.

Teniendo en cuenta los beneficios de los juegos serios en el ámbito de la enseñanza, nuestro propósito con este trabajo es proponer el uso de ProDec [9], un juego serio basado en simulación, para ayudar en la formación del proceso de software.

4 ProDec y el Proceso de Software

ProDec [9] es un juego serio basado en simulación para formar, evaluar y motivar en dirección y gestión de proyectos software, que permite que los alumnos adquieran experiencia práctica en un entorno libre de riesgos, y mejoren sus habilidades como líder de proyecto de cara al mundo laboral.

ProDec permite realizar dos tipos de partida: rápida o completa. En una partida rápida los jugadores pondrán en práctica sus habilidades para la toma de decisiones con el fin de completar con éxito un escenario de proyecto ya definido. Por otro lado, en una partida completa, los jugadores parten desde cero, por lo que tendrán que crear el escenario de proyecto con el que jugarán. El objetivo de ambos tipos de partidas es gestionar con éxito un proyecto software. Esto conlleva completar el proyecto dentro de los plazos de tiempo y costo planificados. Para ello, los jugadores siguen un proceso de tres etapas mediante el cual recorren diferentes etapas del ciclo de vida de un proyecto con el fin de completar el proyecto con éxito. Las tres etapas principales del ciclo de vida de una partida en el juego son: Inicio (E1), Ejecución (E2) y Fin (E3). En el modo de juego de partida completa, la etapa de Inicio se divide, a su vez,

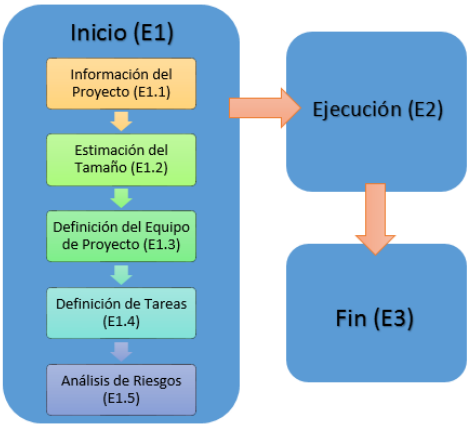


Figura 2. Etapas del Ciclo de vida de una partida.

en cinco sub-etapas secuenciales (véase Figura 2). Los jugadores, a través de estas sub-etapas de ProDec, siguen un proceso que les guía en la creación del plan de proyecto, lo que les permite establecer toda la información necesaria para crear un nuevo plan de proyecto software y poder jugar posteriormente con él, mediante la simulación de su ejecución.

En las siguientes secciones, describimos las características más relevantes de las diferentes etapas del ciclo de vida de una partida en ProDec y su relación con los procesos del ciclo de vida definidos por el estándar ISO/IEC 12207.

4.1 Inicio

La etapa de Inicio (E1), como su nombre indica, es la primera etapa del juego, a través de la cual los jugadores toman, por primera vez, contacto con ProDec. Si los jugadores eligen realizar una partida rápida, estos navegarán por los diferentes escenarios disponibles para el juego, los cuales han sido creados por los profesores, con el fin de analizarlos y evaluar las características de los proyectos presentados. En este caso, los alumnos necesitan conocer los conceptos y técnicas principales de la dirección y gestión de proyectos software y necesitan comprender la información y los datos que el juego suministra, con el fin de que puedan tener en cuenta la dificultad de los distintos escenarios de juego propuestos.

Por otro lado, si los jugadores deciden realizar una partida completa, estos comenzarán un proceso que les guiará en la creación del plan de un proyecto software. Este proceso se compone de cinco sub-etapas secuenciales mediante las cuales, los alumnos podrán suministrar todos los datos necesarios para crear un nuevo plan de proyecto, y tendrán que tomar las decisiones oportunas en todo momento con el fin de planificar el proyecto de la mejor forma posible. Por tanto, durante toda esta etapa los jugadores entran en contacto con el *proceso de planificación de proyecto* (cláusula 6.3.1) y el *proceso de gestión de decisiones* (cláusula 6.3.3) del estándar ISO/IEC 12207.

El *proceso de medición* (cláusula 6.3.7) se cubre con las sub-etapas de Información del Proyecto (E1.1) y Estimación del Tamaño (E1.2), en donde los jugadores proporcionan toda la información necesaria para calcular y establecer el tamaño de los casos de uso, con el fin de realizar la estimación del tamaño del proyecto. Mediante las sub-etapas de Definición del Equipo de Proyecto (E1.3) y Definición de Tareas (E1.4), en donde los jugadores definen los miembros del equipo que realizaran el proyecto y realizan las asignaciones del personal a las tareas, los jugadores toman contacto con el *proceso de gestión de los recursos humanos* (cláusula 6.2.4) del estándar ISO/IEC 12207.

El *proceso de gestión del modelo de ciclo de vida* (cláusula 6.2.1) y todos los procesos involucrados dentro de los *procesos técnicos* (cláusulas 6.4.x), los *procesos de implementación del software* (cláusulas 7.1.x) y los *procesos de apoyo al software* (cláusulas 7.2.x) definidos por el estándar se cubren con la sub-etapa de Definición de Tareas (E1.4), donde los jugadores tienen que definir todas las actividades y tareas necesarias para el proceso de desarrollo del proyecto. En esta etapa, los jugadores tienen que seleccionar el modelo del ciclo de vida que usaran en el desarrollo de

software, así como definir todas las tareas que se realizaran desde el inicio hasta el final del proyecto.

En la última sub-etapa de la etapa Inicio (E1) se realiza el análisis cuantitativo de los riesgos, por lo que en esta etapa los jugadores ponen en práctica sus conocimientos relacionados con el proceso de análisis de riesgos de un proyecto dando soporte, de este modo, al *proceso de gestión de riesgo* (cláusula 6.3.4) del estándar.

4.2 Ejecución

La segunda etapa del ciclo de vida de una partida consiste en la ejecución del proyecto creado o seleccionado en la etapa anterior, es decir, en la etapa de Inicio. Para llevar a cabo esta etapa, ProDec genera automáticamente el código fuente de un archivo donde se encuentran todas las ecuaciones del modelo de simulación de eventos discretos, el cual simula el proyecto descrito en la primera fase del juego. Una vez que el código fuente del modelo de simulación se ha generado, el modelo de simulación se ejecuta y los jugadores pueden comenzar la etapa de control y seguimiento del proyecto.

En esta fase, los alumnos ponen a prueba fundamentalmente dos conceptos de la gestión de proyectos software. Por un lado, ponen en práctica sus conocimientos sobre la técnica de valor conseguido (EVA) con el fin de realizar el seguimiento del proyecto. Por otro lado, los alumnos ponen a prueba sus habilidades en la toma de decisiones frente a los problemas que pueden surgir durante el desarrollo del proyecto con el fin de corregir las posibles desviaciones, que dichos problemas, producen en el desarrollo del proyecto con el objetivo de finalizar el proyecto dentro del tiempo y presupuesto planificados. Entre las decisiones de control y las actividades que los jugadores pueden necesitar realizar, podemos destacar el análisis de los indicadores de seguimiento, el control de los riesgos, y la gestión de los miembros del equipo y de las tareas, donde los jugadores podrán reorganizar el flujo de actividades o contratar/despedir a miembros del equipo de trabajo.

Por tanto, con la etapa de Ejecución (E2), ProDec permite formar en el *proceso de control y evaluación del proyecto* (cláusula 6.3.2) y permite completar la formación en el *proceso de gestión de decisiones* (cláusula 6.3.3), el *proceso de gestión de los recursos humanos* (cláusula 6.2.4), el *proceso de medición* (6.3.7) y el *proceso de gestión de riesgos* (cláusula 6.3.4) del estándar ISO/IEC 12207. En la Figura 3, se resumen los procesos del estándar ISO/IEC 12207 en relación con las etapas del ciclo de vida de una partida de ProDec. Esta relación identifica los procesos del estándar que ProDec permite enseñar a través de sus etapas dentro de una partida de juego.

4.3 Fin

La última etapa consiste en la evaluación de los alumnos. Usando la información que ProDec almacena durante el desarrollo de las partidas y los criterios de evaluación establecidos por los profesores, ProDec genera un informe de evaluación que describe los conocimientos alcanzados por los alumnos con la realización del juego. El informe

generado permite a los jugadores analizar el desarrollo de la partida realizada con el objetivo de aprender de su propia experiencia. Por lo que los alumnos pueden aprender de sus errores y analizar los eventos ocurridos a lo largo del juego para obtener nuevos conocimientos y adquirir más experiencia de cara a partidas futuras.

Por tanto, en la etapa Fin (E3) los jugadores toman contacto con el estándar ISO/IEC 33014 que proporciona una guía informativa sobre la evaluación del proceso

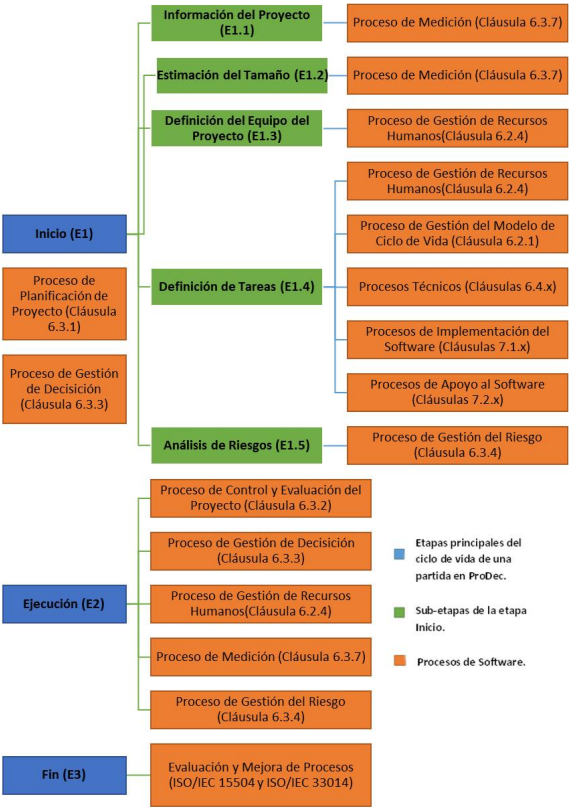


Figura 3. ProDec y los Procesos de Software.

como parte de un marco de trabajo completo y un método para llevar a cabo la mejora de procesos. Esta guía informativa permite una actividad de mejora continua [14]. Además, mediante está última etapa, los jugadores haciendo uso de las lecciones aprendidas, también pueden tomar contacto con el estándar ISO/IEC 15504-4 que proporciona orientación sobre el uso para la mejora de procesos y la determinación de la capacidad del proceso [15].

5 Discusión

La idea de usar ProDec no sólo para formar en dirección y gestión de proyectos software, sino también para apoyar la formación del proceso de software, y su cobertura de los diferentes grupos de procesos del ciclo de vida según el estándar ISO/IEC 12207 ha sido expuesta y evaluada por diez profesores universitarios (n = 10), expertos en el ámbito de la formación en gestión y procesos de la ingeniería del software.

Para realizar dicho estudio se seleccionó el método Delphi, ya que funciona bien cuando se tiene como objetivo mejorar nuestra comprensión de los problemas, oportunidades y soluciones, y cuando la muestra de la población es homogénea y de tamaño pequeño [16].

Las evaluaciones de los expertos concluyen en que el uso de ProDec durante el curso puede ayudar a los alumnos en la adquisición de conocimientos prácticos en el área del proceso de software, obteniendo experiencia práctica con 33 de 43 de los sub-procesos del ciclo de vida del software que el estándar ISO/IEC 12207 define. Por tanto, el uso de ProDec permite formar a los alumnos en el proceso de software, al mismo tiempo que adquieren los conocimientos necesarios de la dirección y gestión de proyectos software. Por otro lado, los expertos están de acuerdo en que ProDec sólo es una herramienta de apoyo a la formación. Esto significa que ProDec ayuda a los alumnos en la aplicación práctica de sus conocimientos y en la adquisición de experiencia en un entorno práctico, pero que necesita que los alumnos tengan conocimientos en los procesos de software antes de ser usado, los cuales deben adquirir durante el curso en las sesiones de teoría.

6 Conclusiones y Trabajos Futuros

La formación en proceso de software es un tema importante a tener en cuenta en los estudios de ingeniería del software con el fin de proporcionar una mejor formación para los futuros ingenieros de software, teniendo en cuenta al mismo tiempo que la adquisición de los conocimientos no sólo sea de forma teórica sino que se ponga especial interés en formar en la práctica profesional. Por tanto, al igual que varios autores, consideramos que el uso de juegos serios y experiencias basadas en simulación nos ayuda a formar en el área de los procesos de software de una forma práctica en un entorno libre de riesgos. ProDec, un juego basado en simulación para formar en dirección y gestión de proyectos software, puede ser usado durante el curso no sólo para formar y poner en práctica los principios de la dirección y gestión de

proyectos software, sino también para formar a los alumnos y profesionales en los procesos del ciclo de vida del software y mejora del proceso de software.

Los resultados de las opiniones de los expertos y el análisis de como ProDec cubre los procesos del ciclo de vida del software del estándar ISO/IEC 12207, nos permite asegurar que el uso de ProDec es útil para los alumnos y les ayuda a consolidar y reforzar sus conocimientos en las áreas de la dirección y gestión de proyecto software y del proceso de software.

Nuestro objetivo es crear una herramienta para dar soporte en la formación de los procesos de software que engloban la dirección y gestión de proyectos software, al mismo tiempo, que sea una herramienta para formar en los procesos del ciclo de vida del software del estándar ISO/IEC 12207. Por este motivo, estamos trabajando en incorporar nuevas características a ProDec en relación con el proceso de software y la dirección y gestión de proyecto software como la gestión de la configuración, la gestión de la calidad, etc. Además, tenemos la intención de integrar ProDec como elemento dentro de un marco de trabajo para el diseño y creación de estrategias de gamificación, con el fin de dar soporte a la formación en proceso de software, y así, promover la participación e incrementar la motivación de los alumnos en esta área del conocimiento.

Agradecimientos

Este trabajo se ha llevado a cabo en el marco de los proyectos del Ministerio de Economía e Innovación TIN2013-46928-C3-2-R, y el proyecto TIC-195 de la Consejería de Economía, Innovación, Ciencia y Empleo de la Junta de Andalucía.

Referencias

[1] W. S. Humphrey, A Discipline for Software Engineering, Boston, MA, USA: Addison-Wesley Longman Publishing Co., 1995.

[2] ISO/IEC, ISO/IEC 15504-5:2012 Information technology -- Process assessment -- Part 5: An exemplar software life cycle process assessment model, 2012.

[3] ISO/IEC, ISO/IEC 12207:2008 - Systems and software engineering — Software life cycle processes, 2008.

[4] A. Baker, E. Oh Navarro and A. Van Der Hoek, "An Experimental card game for teaching software engineering processes.," *Journal of Systems and Software, Software Engineering Education and Training,* pp. 3-16, 2005.

[5] ACM/IEEE-CS Joint Task Force on Computing Curricula, Computer Science Curricula 2013, ACM Press and IEEE Computer Society Press, 2013.

[6] A. Heredia, R. Colombo-Palacios and A. Amescua-Seco, "A Sytematic Mapping Study on Software Process Education," in *Proceedings of the International Workshop on Software Process Education, Training and Professionalism*, Gothenburg, Sweden, 2015.

[7] T. Susi, M. Johannesson and P. Backlund, "Serious games: An overview.," 2007.

[8] ASTD Research, Playing to Win: Gamification and Serious Games in Organizational Learning, 2014.

[9] A. Calderón and M. Ruiz, "ProDec: a serious game for software project management training," in *Proceedings of the 8th International Conference on Software Engineering Advances*, Venice, Italy, 2013.

[10] C. Abt, Serious Games, Lanhan, MD: University Press of America, 2002.

[11] M. Zyda, "From visual simulation to virtual reality to games," *Computer,* vol. 38, no. September, pp. 25-32, 2005.

[12] R. Oliveira Chaves, C. Gresse von Wangenheim, J. C. Costa Furtado, S. Ronaldo Bezerra Oliveira, A. Santos and E. L. Favero, "Experimental Evaluation of a Serious Game for Teaching Software Process Modeling.," *IEEE Transactions on Education.,* vol. 58, no. 4, pp. 289-296, 2015.

[13] U. Aydan, M. Yilmaz and R. V. O'Connor, "Towards a Serious Game to Teach ISO/IEC 12207 Software Lifecycle Process: An interactive learning approach.," in *Software Process Improvement and Capability Determination*, Switzerland, 2015.

[14] ISO/IEC, ISO/IEC TR 33014:2013 Information technology -- Process assessment -- Guide for process improvement, 2013.

[15] ISO/IEC, ISO/IEC 15504-4:2004 Information technology -- Process assessment -- Part 4: Guidance on use for process improvement and process capability determination, 2004.

[16] G. J. Skulmoski, F. T. Hartman and J. Krahn, "The Delhi Method for Graduate Research," *Journal of Information Technology Education,* vol. 6, pp. 1-21, 2007.

AgileRoadmap: Un modelo y estrategia para implantación de prácticas ágiles

Patricio Letelier y Mª Carmen Penadés

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
{letelier, mpenades}@dsic.upv.es

Resumen. La comunidad que trabaja con métodos ágiles no cuenta con un cuerpo de conocimiento consensuado. Si bien esto ha permitido que los métodos ágiles se hayan difundido rápidamente, sin la regulación centralizada de alguna institución, como contrapartida la aplicación de métodos ágiles en muchos casos carece de rigor o su valoración resulta demasiado subjetiva. Claramente existen unos métodos ágiles más populares; Scrum, Kanban, Lean Development y Extreme Programming, cada uno de ellos tiene sus propias fuentes de información y en general, se ha fomentado su aplicación de forma exclusiva. Sin embargo, las prácticas ágiles detrás de dichos métodos son complementarias, e incluso muchas de ellas son comunes a varios métodos. A través de implantaciones reales de métodos ágiles en diferentes empresas hemos ido refinando un enfoque para implantación de agilismo que hemos denominado AgileRoadmap. Nuestra propuesta incluye un modelo y una estrategia para la implantación del agilismo, integrando las prácticas ágiles de dichos métodos más populares y centrándose en la aplicación de prácticas, no en la aplicación de un método en concreto. AgileRoadmap incluye un catálogo de prácticas ágiles y una serie de criterios que ayudan en la selección de prácticas ágiles para un cierto contexto de trabajo. Así, AgileRoadmap puede ayudar a elaborar una hoja de ruta para la implantación del agilismo en equipos de trabajo. En este artículo se presenta AgileRoadmap y una herramienta de apoyo que hemos desarrollado para facilitar su aplicación.

1 Introducción

La gestión ágil de equipos de trabajo es una clara tendencia en la industria. Sin embargo, la implantación del agilismo sigue siendo un gran desafío y el grado de éxito conseguido puede ser muy variable. En la literatura ágil el término más usado para referirse a la aplicación del agilismo en una empresa es "transformación (ágil)" [1] y suele contraponerse a "adopción (ágil)" para enfatizar que lo que se pretende, más allá de cambiar métodos o herramientas de trabajo, es cambiar el comportamiento

y los valores en la empresa. Sin embargo, en nuestra propuesta preferimos utilizar un término menos elegante: "implantación de prácticas ágiles". La experiencia en aplicación real de prácticas ágiles en empresas nos demuestra que si bien la intención de cambio de fondo (transformación) es lo ideal, una estrategia pragmática para conseguirlo normalmente conlleva la aplicación progresiva de prácticas ágiles (no todas a la vez), y probablemente con un aumento gradual de la intensidad de aplicación de la práctica y quizás en convivencia con otras prácticas de ámbito más tradicional. Paralelo a este proceso de aplicación progresiva de prácticas ágiles debe hacerse un esfuerzo adicional en cuanto a potenciar también dicha transformación, no solo destacando los beneficios obtenidos de las prácticas aplicadas, sino también promoviendo que sea el equipo quien asuma el protagonismo en la selección, aplicación y refinamiento de prácticas.

Las dificultades para la implantación de prácticas ágiles quedan en evidencia cuando miramos el resultado conseguido en iniciativas de implantación del agilismo en equipos de trabajo. En muchos casos reales hemos detectado importantes anomalías en dicho resultado. A continuación, se presenta una clasificación que hemos hecho de estas anomalías.

- Implantación "anecdótica". Una frase que ilustra este caso es: "hicimos un proyecto de forma ágil, nos fue bien y hasta fue entretenido, pero luego hemos seguimos trabajando como antes".
- Implantación "inconsciente". En este caso se asegura que se está trabajando de forma ágil (incluso indicando un método ágil específico) pero se tiene muy poca argumentación respecto de qué prácticas se están aplicando y en qué nivel de intensidad, cuáles prácticas explícitamente se han postergado en su aplicación, o cuáles simplemente se han descartado.
- Implantación "limitada". Sucede cuando solo se ha implantado una o muy pocas prácticas ágiles, sin la intención de continuar con la iniciativa. Por ejemplo, muchos equipos se conforman con implantar un proceso iterativo e incremental para autodenominarse ágiles (además es incorrecto afirmar que un proceso iterativo e incremental es una exclusividad de los métodos ágiles).
- Implantación "todo o nada". Se presenta cuando en algunos proyectos se utiliza un enfoque ágil y en otros uno tradicional, de forma totalmente excluyente. Hay muchas prácticas ágiles que podrían aplicarse de forma complementaria en un enfoque tradicional (y viceversa).

Es difícil cambiar radicalmente la forma de trabajo de un equipo, y menos en poco tiempo. Podrían darse condiciones muy favorables tales como: equipo motivado por el cambio, cliente involucrado, dirección de la empresa apoyando la iniciativa, apoyo de un consultor experto en metodologías ágiles (rol Coach o Scrum Master), etc., pero, aun así la formación o preparación del equipo para aplicar una práctica puede requerir bastante esfuerzo. Por ejemplo, la preparación para la práctica “aplicar pruebas automatizadas” (probablemente la práctica más exigente desde el punto de vista técnico) necesita formación del equipo, selección de herramientas, un período de rodaje y ajuste del proceso asociado. Así pues, si bien una estrategia revolucionaria para implantar prácticas ágiles podría ser atractiva en cuanto a conseguir un cambio

muy significativo y en corto tiempo, normalmente no es viable, con lo cual resulta más sensato y eficaz aplicar una estrategia evolutiva, que progresivamente mueva al equipo (y a la organización) hacia el agilismo.

En una estrategia de implantación evolutiva es importante tener una hoja de ruta (roadmap) en la cual se plasmen las decisiones respecto de las practicas ágiles que se desea implantar y en el orden que se hará dicha implantación, considerando estratégicamente los beneficios esperados y el esfuerzo de preparación y puesta en marcha de cada práctica. Una hoja de ruta de implantación de prácticas ágiles se puede ver y gestionar como si fuera un Backlog (una lista de ítems pendientes de ejecutar) que contiene prácticas ágiles como ítems, es decir, es una lista ordenada de mejoras de proceso que se quiere ir implantando a lo largo del tiempo en el equipo de trabajo. Como resultado de nuestra experiencia en consultoría para la implantación de prácticas ágiles en los últimos años hemos ido refinando AgileRoadmap, un enfoque para la implantación de prácticas ágiles, ayudando a establecer y gestionar dicha hoja de ruta. AgileRoadmap tiene los siguientes elementos: un catálogo de 42 práctica ágiles extraídas de los métodos ágiles más populares, un modelo para evaluación y selección de prácticas ágiles que formen una hoja de ruta, y una herramienta de apoyo para facilitar su aplicación.

Hasta donde hemos podido averiguar, actualmente no existe una propuesta que se equipare a AgileRoadmap en cuanto a integrar los elementos antes indicados. Como comentaremos más adelante, las propuestas más próximas son aquellas que recopilan prácticas ágiles pero que se orientan principalmente a realizar una evaluación del nivel de agilismo de un equipo de trabajo.

El objetivo de este trabajo es compartir nuestra experiencia en implantación de agilismo en empresas, experiencia que hemos sintetizado en nuestro enfoque AgileRoadmap. La organización de este artículo coincide con los elementos de nuestro enfoque. En la sección 2 se presenta nuestro catálogo de prácticas ágiles. En la sección 3 se explica el modelo para evaluación y selección de prácticas ágiles. En la sección 4 se describe brevemente la herramienta de apoyo desarrollada para elaborar una hoja de ruta. Finalmente, se presentan las conclusiones y trabajo futuro.

2 AgileRoadmap: Catálogo de prácticas ágiles

Para hacer una implantación de agilismo lo básico como punto de partida es tener conocimiento respecto de agilismo y qué significa ser ágil. La respuesta no es sencilla pues la comunidad ágil no cuenta con un cuerpo de conocimiento consensuado como ocurre, por ejemplo, en gestión tradicional de proyectos donde existe el PMBOK, bajo la supervisión del PMI (Project Management Institute). En el ámbito ágil la única fuente ampliamente reconocida es el Manifiesto Ágil [2], una declaración de los fundamentos y principios generales del agilismo, es decir, con un detalle insuficiente como para llevar a cabo una implantación de agilismo en un equipo de trabajo. Sin embargo, hay una enorme cantidad de información en libros e internet respecto de la interpretación del agilismo y de los métodos ágiles lo cual genera cierta confusión.

En cuanto a recopilación de prácticas ágiles, existen algunas propuestas, las más destacadas se resumen a continuación.

- The Unofficial Scrum Checklist [3]. Propuesta por Henrik Kniberg, está basada en Scrum y es un resumen de su propia experiencia profesional en el campo de la consultoría. Incluye 27 prácticas principales, algunas de ellas descompuestas en otras más específicas.
- SAFe Team Self-Assessment [4]. Propuesta por Dean Leffingwell en el marco del método Scaled Agile Framework (SAFE) [5], incluye elementos de Lean, Scrum y XP. Se evalúan 5 "área de salud" y en cada una de ellas se incluyen ciertas prácticas, que en total son 25.
- Depth of Kanban [5]. Propuesta por Christophe Achouiantz, basada específicamente en el método Kanban. Se establecen 7 dimensiones en las cuales se evalúa la aplicación de ciertas prácticas o aspectos tenidos en cuenta al aplicarlas.

Con respecto de nuestro interés por facilitar la elaboración de una hoja de ruta de prácticas ágiles hacemos las siguientes críticas a los trabajos antes comentados.

- Su propósito es básicamente hacer una evaluación del nivel de agilismo de un equipo, no pretenden apoyar la evaluación o selección de prácticas.
- No incluyen prácticas de todos los métodos más populares.
- La granuralidad de las prácticas en muy gruesa, lo cual dificulta la evaluación de cada práctica pues puede que una práctica sólo se aplique a algunos aspectos.

Cuando comenzamos a participar en implantaciones reales de agilismo en equipos de trabajo inmediatamente vimos necesario contar con un catálogo de prácticas. Necesitábamos hacer una evaluación global del estado de aplicación de prácticas ágiles para luego y consecuentemente establecer recomendaciones respecto de las prácticas que convendría reforzar o comenzar a incorporar. Actualmente tenemos un catálogo de 42 prácticas ágiles provenientes de los métodos ágiles más populares; Kanban [6], Lean Development [7], Scrum [8] y Extreme Programming (XP) [9], y hemos incluido unas pocas de nuestra propia experiencia. 42 prácticas son muchas, pero hemos preferido descomponerlas al máximo para facilitar su evaluación y selección. Por otra parte, vimos interesante plantear las prácticas en términos genéricos, es decir, tanto en su nombre como en su descripción hacerlas legibles para otros contextos (diferentes al de ingeniería de software). El agilismo despierta cada vez más interés en otros ámbitos de trabajo. De hecho, la gran mayoría de prácticas ágiles son directamente aplicables a cualquier contexto de trabajo en equipo (incidencias de infraestructura, tickets de soporte, tramitación de expedientes, etc.).

Nuestra propuesta y catálogo de prácticas promueven la aplicación de prácticas ágiles más que la aplicación de un método ágil específico. En la Fig.1 se ilustra cómo los métodos ágiles tienen algunas prácticas exclusivas y otras comunes con otros métodos, esto evidencia que no sería una buena estrategia el centrarse solo en las prácticas de un método, ignorando las prácticas ofrecidas por otros métodos. Las prácticas ágiles deben ser vistas como oportunidades de mejora que deben ser contrastadas con el contexto del equipo para hacer una adecuada selección.

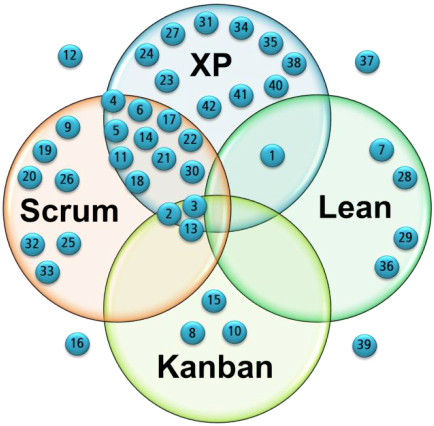


Fig. 1. Prácticas ágiles ofrecidas por los métodos ágiles más populares.

Nos hemos centrado en prácticas "concretas", no se han incluido términos como "Poka Yoke" o similares que, si bien forman parte de la terminología ágil y son interesantes, no consideramos que conlleven directamente acciones específicas para cambiar o introducir nuevas formas de trabajo. De todas maneras, las prácticas ágiles no son recetas directamente aplicables, en general requieren una adaptación al contexto específico. La Tabla 1 muestra la lista de prácticas ágiles de AgileRoadmap. En caso que este documento se esté leyendo como un fichero PDF hemos incluidos enlaces en los códigos de las prácticas, los cuales dirigen a una descripción más detallada en el sitio web de la herramienta de apoyo para AgileRoadmap.

Tabla 1. Catálogo de prácticas ágiles de AgileRoadmap.

#	Práctica ágil	Método del cual proviene
1	Promover la sencillez en todos los aspectos. Ofrecer la solución más simple y mínima que pueda ser satisfactoria para el cliente	Lean, XP
2	Abordar y entregar trabajo terminado de forma incremental	Kanban, XP, Scrum
3	Realizar entregas frecuentes de unidades de trabajo terminadas	Kanban, XP, Scrum
4	Realizar reuniones de planificación frecuentemente (frecuencia de pocas semanas, no meses)	XP, Scrum
5	Acotar el trabajo previsto para un período en base a su estimación y la correspondiente coherencia con la capacidad del equipo	XP, Scrum
6	Organizar el trabajo en iteraciones que agrupan unidades de trabajo que son entregadas en una fecha prevista	XP, Scrum

7	Evitar invertir esfuerzo en adelantar trabajo que no esté comprometido y/o no esté cercano a su entrega	Lean
8	Organizar el trabajo del equipo con el foco en la generación de un buen flujo de trabajo terminado	Kanban
9	Gestión continua y multicriterio del trabajo pendiente para que esté siempre debidamente priorizado	Scrum
10	Limitar el trabajo en proceso (WIP), es decir, la cantidad de unidades de trabajo que tiene el equipo en una determinada actividad	Kanban
11	Formar equipos pequeños y procurar que mantengan sus integrantes	XP, Scrum
12	Acotar el ámbito de trabajo de cada equipo	–
13	Seguimiento continuo (frecuencia de días, no semanas)	Kanban, XP, Scrum
14	Realizar una reunión diaria del equipo al completo, cara a cara y muy breve	XP, Scrum
15	Visualización de todo el trabajo encargado al equipo	Kanban
16	Gestión integrada de todo el trabajo asignado, tanto a nivel del equipo como a nivel de cada miembro	–
17	Cliente en estrecho contacto con el equipo y altamente disponible, incluso si es posible, que esté in-situ	XP, Scrum
18	Que exista una única persona que tome las decisiones respecto de las prioridades del trabajo del equipo y que sea un buen representante de la parte cliente	XP, Scrum
19	Realizar reuniones de revisión del trabajo entregado	Scrum
20	El equipo se auto-organiza y toma las decisiones técnicas	Scrum
21	Jefe de carácter líder y facilitador en lugar de actitud del jefe autoritario y controlador	XP, Scrum
22	Co-localización de los miembros del equipo, todo el equipo trabajando en el mismo espacio físico	XP, Scrum
23	Contar con un espacio físico de trabajo que favorezca la interacción entre los miembros del equipo	XP
24	Establecer y comunicar al equipo la visión del producto o servicio, y reforzarla regularmente	XP
25	Que el equipo sume entre sus miembros las habilidades para abordar todas las actividades necesarias para terminar el trabajo	Scrum
26	Que los integrantes del equipo puedan encargarse de diferentes tipos de actividades (ojalá de todas), aunque puedan ser especialistas en alguna(s) de ellas	Scrum
27	Trabajo centrado en satisfacer pruebas de aceptación acordadas con el cliente	XP
28	Documentar, pero solo lo estrictamente necesario. Que sea rentable el aprovechamiento de la documentación respecto del esfuerzo asociado a elaborarla	Lean
29	Establecer pautas para gestionar convenientemente el re-trabajo	Lean
30	Que exista un líder de mejora de proceso disponible para el equipo	XP, Scrum
31	Establecimiento de estándares para el trabajo técnico del equipo	XP
32	Realizar reuniones de retrospectiva para evaluar el desempeño del equipo y sus formas de trabajo. Mejora continua del proceso	Scrum
33	Acordar y definir qué se entiende por trabajo terminado, tanto para las actividades realizadas por el equipo como respecto de las entregas al cliente	Scrum
34	Trabajo o actividades realizadas en conjunto por dos o más integrantes	XP

35	No abusar de las horas extras, negociar y re-planificar oportunamente para evitarlo	XP
36	Reducir las interrupciones o cambios de contexto que afectan en su trabajo a los miembros del equipo	Lean
37	Establecer una disciplina de aprovechamiento de las reuniones	–
38	Automatizar las pruebas para poder garantizar que el producto mantiene el comportamiento deseado cuando se realizan cambios	XP
39	Postergar hasta último momento la asignación del encargado de realizar una actividad	–
40	Integrar de forma continua en el producto el trabajo terminado	XP
41	Promover que los miembros del equipo en su trabajo lleguen a conocer todas las partes del producto o servicio que han sido encargadas al equipo	XP
42	Mejorar continuamente la organización interna del producto para facilitar su mantenimiento	XP

3 AgileRoadmap: Evaluación y selección de prácticas

El catálogo de prácticas ágiles es necesario para conocer qué prácticas podríamos implantar, pero no es suficiente para hacer una selección acertada de las prácticas con las cuales comenzar la implantación. Esta selección debería estar basada en la evaluación de la conveniencia de cada práctica para el contexto de trabajo del equipo. La Fig.2 muestra el modelo que hemos establecido para ayudar a dicha evaluación.

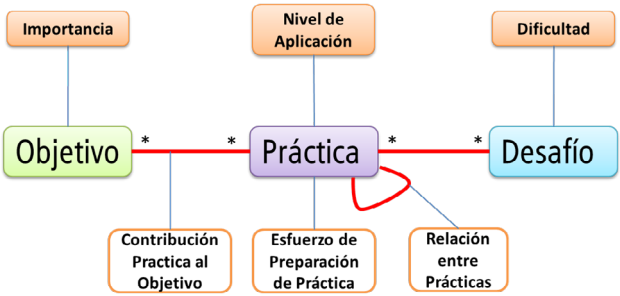


Fig. 2. Prácticas ágiles ofrecidas por los métodos ágiles más populares.

La implantación de prácticas ágiles es una iniciativa de mejora de proceso, y como tal debería estar alineada con ciertos objetivos de mejora. Así pues, debería establecerse una priorización por importancia de los objetivos de mejora que se quieren conseguir. Cada práctica ágil puede contribuir a alcanzar ciertos objetivos de mejora. Así, las prácticas ágiles pueden ordenarse en primera instancia según su contribución a los objetivos más prioritarios. Sin embargo, también debe considerarse

el esfuerzo de preparación para poder aplicar una práctica (p.e. formación en técnicas y herramientas), lo cual podría hacer desistir de su aplicación o decidir postergarla. Además, el nivel de aplicación de una práctica dará un margen mayor o menor de mejora, por ejemplo, si la práctica no está aplicada aportaría un mayor margen de mejora. También habría que considerar las relaciones entre prácticas, ya que una práctica podría verse reforzada por la aplicación de otra práctica o verse inhibida por la ausencia de otra práctica. Existen algunas dependencias evidentes, por ejemplo, aplicar refactoring sin tener un buen soporte de pruebas (y ojalá pruebas automatizadas) puede ser contraproducente. En el método ágil XP Kent Beck enfatiza que mientras más prácticas se apliquen y en mayor profundidad, mejor será el resultado obtenido, pues se produce sinergia entre las prácticas aplicadas. Finalmente, deberían también considerarse los desafíos que suele enfrentar la implantación de una práctica, obstáculos tales como: la relación con el cliente, el contrato, la experiencia del equipo, etc.), esto constituye otra consideración que podría llevarnos a postergar o descartar la implantación de una práctica.

En la Tabla 2 se muestran los objetivos de mejora que hemos establecido. Cada uno de estos objetivos están asociados con prácticas, valorando en cada caso el grado de contribución de la práctica al objetivo.

Tabla 2. Objetivos de mejora utilizados en AgileRoadmap.

Objetivo de mejora	Descripción
Evitar retrasos	Evitar o reducir los retrasos en las entregas.
Gestionar cambios	Gestionar eficazmente los cambios, tanto en los trabajos como en sus prioridades.
Reducir horas extra	Reducir las horas extras o demanda no prevista de recursos humanos adicionales.
Reducir defectos	Reducir defectos en el trabajo entregado al cliente.
Mejorar comunicación	Mejorar la comunicación dentro del equipo y con el cliente.
Involucrar cliente	Involucrar en mayor medida al cliente en la planificación, definición validación del trabajo.
Sistematización trabajo	Mejorar la sistematización del trabajo.
Mejora continua	Promover la mejora continua del proceso empleado por el equipo.
Reducir re-trabajo	Reducir el re-trabajo debido a trabajo defectuoso o incompleto detectado por el equipo.
Time to market	Reducir el tiempo de entrega al cliente, acelerar el "time to market".
Gestión multi-proyecto	Gestionar eficazmente el contexto multi-proyecto.
Visibilidad trabajo	Hacer más visible el trabajo del equipo.
Decisiones oportunas	Tomar decisiones en el momento oportuno.
Gestión RRHH	Mejorar la gestión de recursos humanos en el equipo.
Alineación con negocio	Alineación del trabajo del equipo con los objetivos del negocio.
Evitar sobre-proceso	Evitar costos asociados a la realización de tareas prescindibles o dudosamente rentables.

La Fig.3 muestra la contribución de las prácticas ágiles a cada uno de los objetivos de mejora. En el segmento correspondiente a cada uno de los 16 objetivos se incluyen las prácticas que contribuyen en cierto grado al objetivo. Los círculos concéntricos representan el grado de contribución de las prácticas (de mayor a menor, desde adentro hacia afuera). En la Fig.3 se destaca, por ejemplo, que la práctica 15: "Visualización de todo el trabajo encargado al equipo", es una de las prácticas que más contribuye a varios objetivos. Esta práctica está asociada a la utilización de un tablero kanban o similar donde se puede tener una visualización del estado de todo el trabajo no terminado del cual es responsable el equipo. Otra práctica muy protagonista en cuanto a contribución a varios objetivos es la 9: "Gestión continua y multicriterio del trabajo pendiente para que esté siempre debidamente priorizado", la cual se refiere a la gestión del Backlog, es decir, el trabajo pendiente de ser ejecutado.

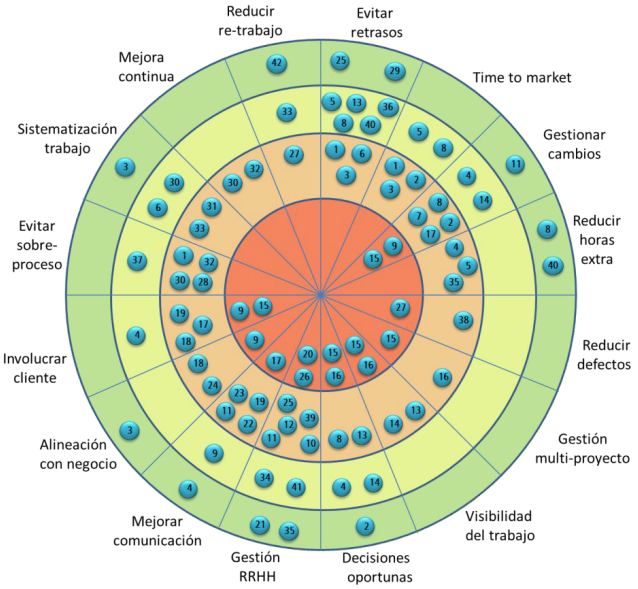


Fig. 3. Contribución de prácticas a objetivos.

4 AgileRoadmap: Herramienta de apoyo

En las primeras implantaciones fuimos utilizando una planilla de cálculo para gestionar la lista de prácticas ágiles y establecer una hoja de ruta para la implantación. No era el soporte más adecuado y decidimos construir una sencilla herramienta para facilitar la evaluación y selección de prácticas. Además, nos interesaba recolectar información respecto de diferentes empresas que estuviesen interesadas en implantar agilismo, por ejemplo, para determinar qué prácticas se eligen como las primeras, cuáles son más postergadas, qué objetivos son los más prioritarios, el grado de aplicación de cada práctica, etc. Desarrollamos un sitio web en que ofrece gratuitamente la gestión de una hoja de ruta para implementación de prácticas ágiles. La dirección del sitio es: http://agileroadmap.tuneupprocess.com. En la Fig.4 se muestra la lista de objetivos de mejora. En dicha interfaz se puede ordenar la lista de objetivos según importancia para el contexto de trabajo. Si se selecciona un objetivo (con el icono del ojo al costado del nombre del objetivo) en la lista de prácticas ágiles mostrada en la Fig.6 se remarcan las prácticas asociadas al objetivo y se muestra también el nivel de contribución de cada una de ellas. Así el usuario puede decidir poner como más prioritaria una práctica arrastrándola a la parte superior de la lista de prácticas. En la Fig.5 se señalan funcionalidades específicas ofrecidas en cada fila de la lista de prácticas.

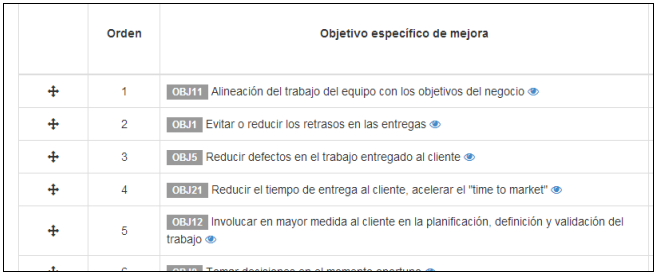


Fig. 4. Lista de objetivos ordenada por importancia en el contexto.

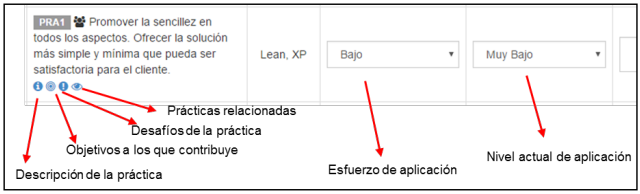
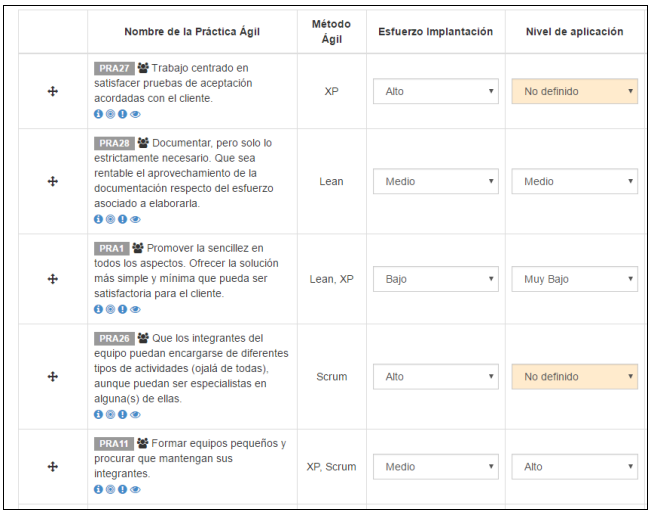


Fig. 5. Detalles de funcionalidad ofrecidos para cada práctica.



	Nombre de la Práctica Ágil	Método Ágil	Esfuerzo Implantación	Nivel de aplicación
✣	PRA27 Trabajo centrado en satisfacer pruebas de aceptación acordadas con el cliente.	XP	Alto	No definido
✣	PRA28 Documentar, pero solo lo estrictamente necesario. Que sea rentable el aprovechamiento de la documentación respecto del esfuerzo asociado a elaborarla.	Lean	Medio	Medio
✣	PRA1 Promover la sencillez en todos los aspectos. Ofrecer la solución más simple y mínima que pueda ser satisfactoria para el cliente.	Lean, XP	Bajo	Muy Bajo
✣	PRA26 Que los integrantes del equipo puedan encargarse de diferentes tipos de actividades (ojalá de todas), aunque puedan ser especialistas en alguna(s) de ellas.	Scrum	Alto	No definido
✣	PRA11 Formar equipos pequeños y procurar que mantengan sus integrantes.	XP, Scrum	Medio	Alto

Fig. 6. Lista de prácticas ágiles ordenada por diferentes criterios.

4.1 Protocolo para la evaluación, selección y aplicación de prácticas ágiles

La hoja de ruta para implantación de prácticas ágiles debería ser el resultado de un diagnóstico del contexto del equipo de trabajo. En nuestra experiencia en implantación de prácticas ágiles hemos ido desarrollando un protocolo para este trabajo de diagnóstico, el cual consta de los siguientes pasos:

1. Seleccionar el equipo de trabajo en el cual se implantarán prácticas ágiles. Se debe realizar la implantación trabajando muy estrechamente con cada equipo. Si bien es necesario hacer un trabajo previo de promoción a niveles de supervisión o dirección para conseguir el apoyo para la iniciativa, el trabajo de implantación debe hacerse con el equipo, no debería ser visto como una imposición desde "arriba", ni dejarse a la motivación y esfuerzo heroico de algunos integrantes del equipo.
2. Estudiar globalmente el trabajo encargado al equipo y prestar atención a su posible diversidad. Lo más sencillo sería que el equipo sólo trabajase en una línea de trabajo (producto, servicio o proyecto), pero desafortunadamente es frecuente que un equipo sea responsable de varias líneas de trabajo a la vez. Es muy importante que si las líneas de trabajo tienen características distintas se adapte el

proceso a sus necesidades, es decir, que se apliquen o no ciertas prácticas. Incluso en un extremo podría llegar a plantearse la conveniencia de tener diferentes hojas de ruta para diferentes líneas de trabajo del equipo. Por ejemplo, sería un error que en todas las líneas de trabajo del equipo se estime y planifique usando Sprints, siendo que, en alguna línea de trabajo la demanda no es previsible, y/o no se desea agrupar el trabajo en Sprints. Estas mezclas se presentan de forma natural cuando ya se ha hecho una entrega del producto al cliente y posteriormente se trabaja en cambios de cierta envergadura (p.e. nuevos módulos o integraciones), pero al mismo tiempo se debe estar ofreciendo un mantenimiento y soporte continuo.

3. Establecer los objetivos que pretende la iniciativa de mejora de proceso. Este paso obliga a una reflexión respecto del desempeño del equipo en el contexto de sus líneas de trabajo. Cada práctica ágil contribuye en cierta medida a unos objetivos con lo cual, evaluando la importancia de los objetivos en el contexto del equipo de trabajo se seleccionarán prácticas candidatas, aquellas que contribuyan a dichos objetivos. Se recomienda centrarse en un reducido número de objetivos pues de todas formas probablemente no será posible implantar demasiadas prácticas ágiles a la vez, y menos cuando se trata de una primera implantación.
4. Establecer el nivel de aplicación actual de cada práctica candidata. Es importante evaluar si una práctica está ya aplicada, no aplicada, parcialmente aplicada o simplemente determinar que no interesa aplicarla. Una práctica actualmente no aplicada podría tener un mayor efecto que una práctica que ya se está aplicando de forma parcial. Es decir, una práctica no aplicada o aplicada parcialmente ofrece un mayor margen de mejora respecto de la situación actual.
5. Establecer el nivel de dificultad que tendrían los desafíos de implantación de cada práctica. Hay que tener presente los desafíos que podría enfrentar una práctica para que sea viable su aplicación. A veces factores difíciles de modificar, tales como la relación con el cliente, el contrato, la cultura organizacional, etc. Pueden llevarnos a descartar o postergar la aplicación de una práctica.
6. Evaluar la aplicabilidad de cada práctica. Con toda la información recopilada en los pasos anteriores puede establecerse una valoración de la aplicabilidad de la práctica, siguiendo las siguientes directrices: valorar positivamente la importancia de los objetivos a los que contribuye la práctica, valorar positivamente el nivel de aplicación actual de la práctica (mayor mientras menos aplicada esté la práctica), valorar el nivel de dificultad de los desafíos de la práctica (mientras menos mejor), valorar el esfuerzo de aplicación de la práctica (mientras menos esfuerzo mejor). Revisar las relaciones entre prácticas, especialmente entre las candidatas y otras que pudiesen haber quedado excluidas de este conjunto, considerando la posible incorporación de alguna práctica que pudiese reforzar la aplicación de una práctica candidata. La aplicación de todas estas pautas provocaría cambios en el ordenamiento de la lista de prácticas candidatas para implementar.
7. Seleccionar un conjunto de entre las prácticas candidatas y formar al equipo en su aplicación. Para algunas prácticas podría también ser necesaria una selección de herramientas de apoyo y una correspondiente formación en su uso. La implantación de prácticas ágiles debería, consecuentemente, ser ágil. Nuestra recomendación es seleccionar un primer conjunto de prácticas ágiles que sea

factible de preparar en no más de un mes. Una vez puestas en marcha, debería hacerse un estrecho seguimiento para supervisar cómo están actuando las prácticas, si es necesario hacer alguna adaptación, o incluso si una práctica no nos satisface podríamos quitarla. En la medida que se consolide la aplicación de ciertas prácticas podemos plantearnos la incorporación de nuevas prácticas repitiendo el protocolo antes comentado, o directamente cogiendo prácticas que ya habían sido evaluadas y postergadas para una posterior aplicación.

El protocolo indicado se aplicaría en base a reuniones en las cuales los involucrados irían tomando decisiones asociadas a cada uno de estos pasos. En la herramienta se iría registrando la información establecida para cada práctica, y las priorizaciones asignadas a objetivos y prácticas.

5 Conclusiones

Nuestro enfoque AgileRoadmap ha nacido y se ha ido refinando en el marco de nuestra participación en consultoría para implantación de agilismo en equipos de trabajo. En los últimos años hemos participado en más de 10 implantaciones reales, además de otras muchas asesorías y formaciones asociadas a métodos ágiles en empresas. Muchas lecciones aprendidas las hemos ido integrando en nuestra propuesta AgileRoadmap, algunas de ellas son:

- La implantación del agilismo no consiste en conseguir aplicar un método ágil u otro, sino que debe enfocarse en aplicar prácticas ágiles, mientras más y en mayor profundidad más significativa será la mejora obtenida.
- Una estrategia de implantación evolutiva es más realista y factible de llevar a cabo. Pero exige un seguimiento muy estrecho para resolver cuestiones relativas al ajuste de la intensidad de aplicación de las prácticas ágiles, la probable convivencia con ciertas prácticas de carácter tradicional, y las dudas o cuestionamientos que seguramente se presentarán.
- Es imprescindible una evaluación y selección de prácticas considerando las características del contexto de trabajo del equipo, y prestando especial atención a las características de cada línea de trabajo encargadas a un mismo equipo. El modelo de AgileRoadmap incorpora suficientes elementos para realizar un diagnóstico exhaustivo del contexto de trabajo del equipo y con ello seleccionar las prácticas que progresivamente se implantarán. La priorización de objetivos de mejora desempeña un papel clave para guiar la selección de prácticas ágiles que se implantarán en un momento dado. En este sentido AgileRoadmap ofrece una lista de 16 objetivos, los cuales tienes asociadas determinadas prácticas ágiles que en alguna medida contribuyen a lograrlos.
- Una implantación ágil debe ser por naturaleza también ágil. La hoja de ruta de implantación de prácticas ágiles debe gestionarse como un Backlog de mejora. Sugerimos no retrasar la puesta en marcha de una iniciativa que comience el camino del equipo hacia el agilismo, y en no más de un mes comenzar a aplicar con cierta profundidad un conjunto reducido de prácticas, acotado principalmente por el hecho que no sea necesario más de un mes para su preparación.

Después de año y medio desde que publicamos el sitio web AgileRoadmap tenemos registrados 197 hojas de ruta de usuarios (equipos) de diversos países (las estadísticas recolectadas están disponibles en el mismo sitio). En general, los datos no nos han sorprendido, más bien confirman lo que esperábamos. Por ejemplo, los objetivos señalados como más importantes son: "Evitar o reducir retrasos en las entregas", "Alineación del trabajo del equipo con los objetivos de la empresa", y "Reducir los defectos en el trabajo entregado al cliente". Las prácticas favoritas para ser implantadas antes son: "Promover la sencillez" (la solución más simple que sea satisfactoria para el cliente), "Abordar y entregar el trabajo de forma incremental", y "Realizar entregas frecuentes". Las prácticas con mayor nivel de aplicación son: "Co-localización del equipo" y "Contar con un espacio físico de trabajo que favorezca la colaboración". Por contraparte, las prácticas menos aplicadas son: "Automatización de las pruebas" y "Mejora interna del producto" (*refactoring*).

Actualmente estamos trabajando en mecanismos para generación automática de una hoja de ruta. El usuario respondería un cuestionario para la evaluación de prácticas ágiles y al finalizar se le generaría automáticamente una hoja de ruta a modo de sugerencia. Respecto de la información recolectada a través de la herramienta tenemos pendiente establecer mecanismos que nos permitan conseguir retro-alimentación respecto del grado satisfacción de los usuarios, así como información para refinar el modelo subyacente con nuevos elementos o relaciones que puedan mejorarlo. Finalmente, queremos ofrecer más pautas respecto de cómo llevar a cabo el diagnóstico y elaboración del roadmap, sugiriendo un protocolo de reuniones, participantes e hitos del trabajo asociado.

Referencias

1. Sahota M. An Agile Adoption and Transformation Survival Guide: Working with Organizational Culture. InfoQ Enterprise Software Development Series. 2012.
2. Varios autores. Manifiesto for Agile Software Development. http://www.agilemanifesto.org/.
3. Kniberg H. The Unofficial Scrum Chacklist. https://www.crisp.se/wp-content/uploads/2012/05/Scrum-checklist.pdf .
4. Leffingwell D. SAFe Team Self-Assessment. http://www.scaledagileframework.com/metrics/#T4.
5. Achouiantz C. Depth of Kanban. Blog Lean-Agile Software Development. http://leanagileprojects.blogspot.com.es/2013/03/depth-of-kanban-good-coaching-tool.html.
6. Anderson D.J. Kanban: Successful Evolutionary Change for Your Technology Bussines. Blue Hole Press, 2010.
7. Poppendiek M, Poppendiek T. Leading Lean Software Development. Addison-Wesley, 2010.
8. Schwaber K., Sutherland J. The Scrum Guide. http://www.scrumguides.org/
9. Beck K. Extreme Programming Explained: Embrace Change. Addison-Wesley, 1999.

Categorización de actividades de seguridad en el desarrollo de software

José C. Sancho[1], Andrés Caro[2], Pablo García[2], Ángel Quesada[3]

[1] Viewnext S.A., Cátedra de Viewnext-UEx "Seguridad y Auditoría de Sistemas Software", Parque Científico y Tecnológico, C/Avda. de la Universidad s/n, 10071, Cáceres, España

[2] Dpto. Ingeniería de Sistemas Informáticos y Telemáticos, Escuela Politécnica, Universidad de Extremadura, C/Avda. de la Universidad, s/n. 10003 Cáceres, España

[3] Viewnext S.A., Centro CENIT Cáceres, Parque Científico y Tecnológico, C/Avda. de la Universidad s/n, 10071, Cáceres, España

{jcsanchon, andresc, pablogr}@unex.es / aquesada@viewnext.com

Resumen. Resulta evidente la necesidad de considerar la seguridad en todas las tareas del ciclo de vida del software. Su inclusión desde las fases más tempranas de desarrollo evitará futuros sobrecostes en las fases finales, que suelen ser las más complejas. La detección de fallos de seguridad o vulnerabilidades en las fases iniciales de desarrollo garantizan la reducción de posibles ataques maliciosos, además de contribuir a la reputación corporativa de las empresas que desarrollan software. En el camino hacia particularizar un modelo de desarrollo de software seguro por defecto, y tras analizar diferentes marcos de trabajo, se han identificado una serie de prácticas de seguridad recurrentes en todos ellos, y, por ello, fundamentales. Las carencias detectadas en los modelos estudiados se subsanan añadiendo nuevas actividades de seguridad, propuestas por los autores. Este trabajo realiza una categorización de las actividades de seguridad del ciclo de vida del software, de manera que se encuentren correctamente planificadas y se implementen de forma sistemática, garantizando una mayor seguridad del software desarrollado.

Palabras Clave: Modelo de Seguridad, Software Seguro, Ciclo de Vida, Desarrollo Seguro.

1 Introducción

La aparición de multitud de vulnerabilidades y el aumento de ataques a los sistemas de información, unido a la integración de internet en los procesos de negocio de las empresas, hacen de la seguridad en el desarrollo del software un factor crítico. El último informe publicado por el Centro Criptológico Nacional (CCN-CERT) sobre ciberamenazas y tendencias, prevé un incremento del 40% en los ciberataques a la Administración y a empresas de interés estratégico [1].

Los procesos de desarrollo de software habituales se orientan exclusivamente a la funcionalidad. De este modo, las prácticas de seguridad no son tenidas en cuenta durante la construcción del software, por lo que los fallos y vulnerabilidades se detectan y solventan en las fases finales de desarrollo, las más complejas y que acumulan más retrasos. Ello provoca efectos perniciosos, como el aumento y desajuste producido por los altos costes de resolución de vulnerabilidades (*deuda técnica*) y la pérdida de reputación de la empresa desarrolladora.

Este trabajo se fundamenta en el interés en temas de *ciberseguridad* en los últimos años en nuestra comunidad [2]. El objetivo se centra en categorizar las actividades de seguridad, identificadas en [3], consideradas esenciales y adecuadas. Estas actividades integran y garantizan la seguridad en todas las fases de construcción del software, permitiendo particularizar un modelo propio de desarrollo de software seguro.

2 Análisis de metodologías de desarrollo seguro

Para suplir la omisión de actividades de seguridad e integrar esta carencia en el ciclo de vida de desarrollo de software, emergen diversos marcos de trabajo especializados, analizados en [3]. Aunque existen normas y propuestas que pueden complementarlas, y que están más relacionadas con la Ingeniería del Software, para su aplicación práctica en una compañía como ViewNext, las aquí estudiadas resultan, inicialmente, suficientes. Así, pueden citarse el Microsoft Security Development Lifecycle (Microsoft SDL) [4], Oracle Software Security Assurance (OSSA) [5], Comprehensive Lightweight Application Security Process (CLASP) de Open Web Application Security Project (OWASP) [6], Team Software Process Secure (TSP-Secure) [7], y Software Assurance Maturity Model (OpenSAMM) de OWASP [8].

Entre estos modelos se identifican actividades de seguridad comunes a varias de ellas. A juicio de los autores, estas prácticas deberían integrarse en cualquier modelo de desarrollo seguro de software, para reducir las posibilidades de ataques o intrusiones, y permitir una eficiente resolución de vulnerabilidades durante el proceso de construcción del software. Las actividades identificadas son:

- Plan estratégico unificado que avale el nivel de validación de seguridad del software desarrollado: *Estrategia y Orientación*.
- Formación en materia de seguridad de los miembros de los grupos implicados en todo el proceso de desarrollo del software: *Formación*.
- Identificación y definición de los riesgos específicos que correspondan al modelo de negocio del cliente: *Definición de riesgos*.
- Obtención y validación de los requisitos de seguridad: *Validación de Requisitos*.
- Análisis y modelado de amenazas que identifiquen y aseguren la superficie de ataques maliciosos al sistema: *Modelado de Amenazas*.
- Revisión de la seguridad del diseño: *Revisión del Diseño*.
- Revisión del código fuente desarrollado, a través de un análisis estático y dinámico del código: *Revisión del Desarrollo*.
- Pruebas de seguridad que verifican calidad y seguridad: *Testing de Seguridad*.
- Validación de la seguridad de las salidas del código: *Validación de Salidas*.

- Implantación de un plan de respuesta a incidentes en la detección y resolución de nuevas vulnerabilidades: *Plan de Respuesta a Incidentes*.

Sin embargo, pese a la madurez que poseen los modelos estudiados, se detectan una serie de carencias que han sido subsanadas y corregidas, añadiendo tres novedosas actividades de seguridad. Estas actividades adicionales tienen como objetivo fundamental mejorar y ampliar la seguridad en el desarrollo.

- *Observatorio de Seguridad:* Actividad de vigilancia tecnológica que detecta la aparición de nuevas vulnerabilidades y posibles ataques al software.
- *Repositorio de vulnerabilidades:* Retroalimentación empírica de la base de conocimiento de vulnerabilidades, permitiendo conocer su estado, analizar su evolución y criticidad. Su objetivo principal es transformar las actuaciones reactivas aplicadas en fases avanzadas de un proyecto de software, en acciones preventivas adoptadas en las fases tempranas de desarrollo.
- *Estado del proyecto:* Revisión, cuantificación y clasificación de las vulnerabilidades, en cualquier instante del desarrollo de software. Garantiza el cumplimiento de las directrices de seguridad propuestas.

3 Categorización de actividades de seguridad en el desarrollo.

Las metodologías de desarrollo seguro *Microsoft SDL* [4] y *SAMM* [8], categorizan las prácticas o procedimientos de seguridad: la primera de ellas, ordena los procedimientos seguros, según su ejecución en las fases del ciclo de vida del software tradicional; *SAMM* [8] encuadra sus doce prácticas de seguridad, alrededor de cuatro funciones de negocio: *Gobierno, Construcción, Verificación e Implementación*.

La propuesta presentada en este trabajo sigue la analogía de *SAMM*, organizando las actividades de seguridad de la metodología sistemática propia en cuatro *áreas de desarrollo*, todas integradas en los procesos y actividades del desarrollo de software. A continuación, se cita su nombre, la definición global de los procesos que delimita y las actividades de seguridad, listadas en el apartado anterior, que ejecutan.

- *Políticas*: Área de desarrollo orientada principalmente a la definición de los objetivos y las directrices globales en materia de seguridad del desarrollo de software. Las prácticas de seguridad de esta categoría incluyen la activa participación de todos los grupos implicados, con la finalidad de unificar la seguridad del software en la organización y proteger el software desarrollado. Estas prácticas son: *Estrategia y Orientación, Formación* y *Definición de riesgos*.
- *Metodología SDL*: Enfocada exclusivamente a los procesos y actividades relacionados con la construcción de software seguro por defecto. Las actividades de seguridad son: *Validación de Requisitos*, *Modelado de Amenazas, Revisión del Diseño, Revisión del Desarrollo, Testing de Seguridad* y *Validación de Salidas*.
- *Supervisión*: Ejecuta los procesos y actividades de revisión de vulnerabilidades e incidencias operativas, realizando la evaluación final del funcionamiento y

seguridad del software entregado. *Estado del Proyecto* y *Evaluación y métricas* se encuadran en esta categoría.

- *Observatorio*: Dirigida a procesos y actividades de continua vigilancia con la finalidad, de detectar la aparición de vulnerabilidades desconocidas hasta el momento y abrir nuevas líneas de investigación sobre innovadoras técnicas de ataques desconocidas. Estas prácticas son: *Observatorio de Seguridad, Repositorio de vulnerabilidades* y *Plan de Respuesta a Incidentes*.

4 Conclusiones y futuros trabajos

Se evidencia que una de las grandes necesidades detectadas, actualmente, para garantizar la mayor seguridad del software desarrollado, es incorporar un conjunto de actividades de seguridad, correctamente categorizadas, planificadas y ejecutadas de forma sistemática, dentro de las fases del ciclo de vida del software.

Los futuros trabajos orientan todos sus esfuerzos hacia la construcción y particularización de un modelo propio de desarrollo de software seguro por defecto.

Agradecimientos.

Los autores agradecen la financiación recibida por parte de la Junta de Extremadura (Consejería de Economía, Comercio e Innovación), FEDER (Fondo Europeo de Desarrollo Regional: GR15173) y Cátedra ViewNext-UEx.

Referencias

1. Centro Criptológico Nacional Computer Emergency Response Team: Informe Anual - Ciberamenazas 2015 - Tendencias 2016. *Web*. https://www.ccn-cert.cni.es/documentos-publicos/comunicados/1486-np14ia-informe-anual/file.html.(2016). Accedido: 20/04/2016.
2. Caro, A.: Una apuesta por la educación en ciberseguridad desde el ámbito universitario. *Jornadas Univ. de Investigación en Ciberseguridad (JNIC 2015)*, pp. 198-202, 2015.
3. Sancho, J. C.; Caro, A.; García, P.: Análisis de metodologías de Desarrollo de Software Seguro. *Jornadas Univ. de Investigación en Ciberseguridad (JNIC 2016)*, pp.42-47, 2016.
4. Lipner, S.; Howard, M.: El ciclo de vida de desarrollo de seguridad de Trustworthy Computing. *Unidad tecnológica y empresarial de seguridad-Microsoft Corporation, 2004 Annual Computer Security Applications Conference, copatrocinada por IEEE*. (2004).
5. Redwood Shores, C.A.; Oracle Corporation: *Oracle Corporation Web*. https://www.oracle.com/support/assurance/index.html. Accedido: 20/04/2016.
6. Open Web Application Security Project: Comprehensive Lightweight Application Security Process. *OWASP Web*. https://www.owasp.org/index.php/CLASP_Concepts. Accedido: 20/04/2016.
7. Noopur, D.; Philip, L.; Miller, W. R.; Nichols, R. C.: TSP Secure. *Proceedings of the Fourth Annual TSP Symposium*, pp. 3-8, (2009).
8. Open Web Application Security Project: Software Assurance Maturity Model. *OpenSAMM Web*. http://www.opensamm.org. Accedido el 20/04/2016.

Simulación para la Toma de Decisiones en la Gestión del Proceso de Evaluación de la Usabilidad

Nuria Hurtado[1], Mercedes Ruiz[1], Elena Orta[1], Jesús Torres[2]

[1]Departamento de Ingeniería Informática
Universidad de Cádiz (España)
Avenida Universidad de Cádiz nº 10 11519 – Puerto Real - Cádiz (España)
(Teléfono: +34 956 483441) (fax: +34 956 015139)
[2]Departamento de Lenguajes y Sistemas Informáticos
Universidad de Sevilla (España)
Avda. Reina Mercedes s/n - 41012 - Sevilla (España)
(Teléfono: +34 95 4552769) (fax: 95 4557139)

{nuria.hurtado, mercedes.ruiz, elena.orta}@uca.es
jtorres@lsi.us.es

Resumen.

Este artículo está desarrollado en el contexto de la Ingeniería de la Usabilidad, más concretamente, se centra en el uso de las técnicas de modelado y simulación para la ayuda a la toma de decisiones en el ámbito de la evaluación de la usabilidad. El principal objetivo del artículo es presentar UESim: un modelo de simulación basado en Dinámica de Sistemas para ayuda a la toma de decisiones en la configuración del equipo de evaluadores durante el proceso de evaluación de la usabilidad. Para desarrollar esta investigación se han seguido cuatro fases: a) identificación b) desarrollo c) ejecución y finalmente, d) reflexión. En relación con estas fases el artículo describe en primer lugar la revisión de la literatura, en segundo lugar la construcción del modelo y su validación, posteriormente la simulación y análisis de los resultados y por último la reflexión sobre los mismos. A través de tres casos de estudio diferentes se analizan los efectos de diferentes composiciones del equipo de evaluación en los resultados de la misma. Los resultados de la simulación muestran la utilidad del modelo bajo diferentes configuraciones variando el número y experiencia de los evaluadores. Una de las principales ventajas del modelo es que permite a los desarrolladores observar la evolución de los indicadores clave del proceso de evaluación a lo largo del tiempo. UESim representa una herramienta parametrizable de ayuda la toma de decisiones en la gestión del proceso de evaluación de la usabilidad, ya que, es posible analizar cómo diferentes decisiones de gestión afectan a los indicadores clave del proceso: tiempo, coste y número de problemas de usabilidad encontrados.

Palabras clave: Evaluación de la Usabilidad, Toma de Decisiones, Modelado y Simulación, Mejora del Proceso Software, Gestión de Procesos.

Evaluación de Juegos Serios: una Revisión Sistemática de la Literatura con Aplicación en Dirección y Gestión de Proyectos Software

Alejandro Calderón [1] y Mercedes Ruiz [1]

[1] Departamento de Ingeniería Informática, Universidad de Cádiz.
Avenida de la Universidad de Cádiz, 10
11519 - Puerto Real (Cádiz), España.
{alejandro.calderon, mercedes.ruiz}@uca.es

Abstract. La formación que reciben los futuros profesionales en dirección y gestión de proyectos software es un tema de gran relevancia a tener en cuenta en el sector de la educación. La revisión sistemática de la literatura que se presenta en este trabajo persigue analizar y resumir el estado del arte actual de los diferentes métodos y procedimientos usados para evaluar juegos serios. Para llevar a cabo la revisión se definió un procedimiento basado en búsquedas automáticas en diferentes bases de datos digitales de reconocimiento científico (SCOPUS, Web of Science, IEEE Xplore, SpringerLink, ACM Digital Library). Dichas búsquedas automáticas permitieron encontrar un total de 1199 trabajos relacionados con la evaluación de juegos serios de los cuales 102 fueron seleccionados como estudios primarios. El proceso de revisión llevado a cabo fue complementado con la realización de búsquedas manuales en la literatura gris. Nuestra revisión sistemática de la literatura identifica los principales métodos utilizados para evaluar juegos serios, así como, los ámbitos de aplicación en el que la evaluación tiene lugar, los tipos de juegos serios que se evalúan y las principales características para evaluar la efectividad educacional de los juegos serios. Además, nuestro trabajo también analiza los procedimientos que los autores suelen seguir para realizar la evaluación de los juegos serios y el tamaño de la población que participa en dichas evaluaciones. Los resultados obtenidos con este trabajo son útiles para los investigadores y profesionales que persiguen evaluar los juegos serios en diferentes campos, pero especialmente para aquellos interesados en la evaluación de juegos serios en el ámbito de la dirección y gestión de proyectos software.

Palabras claves: Juegos Serios, Evaluación, Revisión Sistemática de la Literatura, Dirección y Gestión de Proyectos Software

Software Project Management: Learning from our mistakes

Pedro Silva[1], Ana M. Moreno[1], Lawrence Peters[2]
[1] Computer Science School. Universidad Poltécnica de Madrid
{pedro.silva@upm.es; ammoreno@fi.upm.es)
[2] *Software Consultants International Limited, U.S.A.*
peters@gmail.com

Abstract. While it is important to know what software project managers should do to make project success more likely, a complementary question is, What should they not do? In other words, what practices should software project managers avoid to make success more likely? To answer this question we performed an extensive literature search looking for information about software project management antipatterns over the last 10 years. We generated a consolidated list of antipatterns and categorized them according to different criteria: impacted software project management activity, impacted role, root cause and solution type. The paper discuss the implications of such categorization and provides some tips to deal with software project management antipatters.

IEEE Software, vol. 32(3), May/June 2015. DOI: 10.1109/MS.2015.71

Tema Abierto

Tema Abierto
Coordinador: Jesús J. García Molina

J. David Patón-Romero y Mario Piattini. *Auditorías de Green in IT: Un Mapeo Sistemático.* (Completo)

Ana Gabriela Núñez Ávila, Mª Carmen Penadés Gramage y José H. Canós Cerda. *Herramienta de Soporte a la Evaluación y Mejora de la Gestión de Planes de Emergencia.* (Completo)

Aurora Vizcaíno, David Valencia, Juan Pablo Soto, Lilia García-Mundo y Mario Piattini. *¿Qué desafíos presenta el desarrollo global del software? Aprende jugando.* (Corto)

Enrique Moguel, Juan Carlos Preciado, Fernando Sánchez-Figueroa y Juan Hernández. *Smart Spaces: sistema de tecnoinclusión inteligente.* (Corto)

Auditorías de *Green in IT*: Un Mapeo Sistemático

J. David Patón-Romero, Mario Piattini

Grupo de Investigación Alarcos, Instituto de Tecnologías y Sistemas de Información,
Universidad de Castilla-La Mancha, Paseo de la Universidad, 4,
13071 Ciudad Real, España
{JDavid.Paton, Mario.Piattini}@uclm.es

Resumen. En los últimos años el mundo ha ido experimentando una serie de cambios ambientales que han hecho que en la sociedad surja una fuerte convicción en pos de proteger el medioambiente. Las Tecnologías de la Información (TI) y, de manera especial, las tecnologías software, pueden contribuir a la ecosostenibilidad de dos maneras: "Green By IT", en el sentido de que las TI pueden proporcionar herramientas que permitan llevar a cabo tareas de una manera adecuada para el medioambiente, y "Green In IT", cuando las propias TI tienen impacto en el medioambiente, debido a su consumo energético. Sin embargo, las técnicas de Green in IT son relativamente jóvenes y no existe ningún estándar o marco que permita controlar su correcta implementación y/o funcionamiento. Por ello, el objetivo del presente mapeo sistemático es recopilar el conocimiento actual en relación a las auditorías de Green in IT, con el fin de poder determinar cuáles son las características más importantes a la hora de desarrollar un marco de auditoría de Green in IT. Los resultados obtenidos demuestran la novedad de esta área y la casi nula existencia de estudios relacionados con ésta, y, por ello, la necesidad de elaborar un marco de auditoría de Green in IT.

Palabras clave: Auditoría, Green in IT, Mapeo sistemático.

1 Introducción

Los vertiginosos cambios, mucho más allá del cambio climático, que lleva sufriendo el planeta los últimos 30 años han hecho que la sociedad y, sobre todo, las organizaciones se replanteen en muchos aspectos la eficiencia, eficacia y consumo de sus actividades, con el fin de reducir el impacto que provocan en el medioambiente.

A la hora de incrementar la sostenibilidad, el impacto de la tecnología es importante desde dos puntos de vista diferentes [3]. Por un lado, la tecnología ayuda a las organizaciones a abordar las cuestiones medioambientales (reuniones virtuales, desmaterialización de actividades, mejoras en logística, sistemas de transporte inteligentes, *smart grids*, etc.); mientras que, por otro, la tecnología en sí misma es responsable de una importante degradación medioambiental (por ejemplo, la cantidad de energía consumida por los procesos de ingeniería utilizados para fabricar los productos tecnológicos).

En este sentido, Erdélyi [4] destaca que las Tecnologías de la Información (TI de ahora en adelante) pueden contribuir a la ecosostenibilidad de dos maneras: "Green By

IT" en el sentido de que las TI pueden proporcionar herramientas que permitan llevar a cabo tareas de una manera adecuada para el medioambiente (es decir, TI como habilitador (*enabler*) en el sentido de Unhelkar [10]), y "Green In IT", cuando las propias TI tienen impacto en el medioambiente, por el consumo energético y las emisiones producidas por los propios elementos de las TI (es decir, TI como productor). Se suele hablar de "Green IT" para referirse a la combinación de Green by IT y Green in IT.

Este estudio se centra dentro del Green in IT, y más concretamente en identificar los estándares o marcos que permitan controlar la correcta implementación y/o funcionamiento del Green in IT, y muy especialmente su auditoría. Dentro de Green in IT se ha abordado en diferentes proyectos la reducción de consumo energético en la computación en la nube, en el hardware, en los centros de procesos de datos, etc., pero no su auditoría [2].

Desde el punto de vista de la auditoría, dentro de las TI ya existe un marco de referencia para auditores desarrollado por ISACA (*Information Systems Audit and Control Association*) llamado COBIT 5 (*Control Objectives for Information and related Technology*) [11]. Si bien es cierto que este marco ofrece una visión bastante amplia de todo lo relacionado con las TI, e incluso hay versiones adaptadas a ámbitos específicos como la seguridad, COBIT aún no cuenta con ninguna versión ni ningún mecanismo de control relacionado con el Green IT, y menos aún con el Green in IT.

Así pues, creemos que puede resultar muy útil un mapeo sistemático que otorgue el conocimiento más actual posible acerca de esta área de investigación, con el fin de establecer los pilares sobre los que poder desarrollar técnicas y/o marcos de auditoría de Green in IT.

El resto de este estudio está organizado de la siguiente manera: en la Sección 2 se describe el protocolo de investigación llevado a cabo para realizar el mapeo sistemático. La Sección 3 contiene los resultados obtenidos de dicho mapeo. En la Sección 4 se discuten las observaciones principales de los resultados, así como las limitaciones e implicaciones de este campo. Y, finalmente, en la Sección 5 se presentan las conclusiones y el trabajo futuro a realizar en el campo de las auditorías de Green in IT.

2 Protocolo de Investigación

Un mapeo sistemático es un método para recopilar y categorizar la información existente acerca de un tema de investigación. El presente mapeo sistemático se ha llevado a cabo siguiendo las líneas/guías provistas en trabajos como Genero et al. [7], Budgen et al. [1], Petersen et al. [9], and Kitchenham [8].

El mapeo sistemático se ha realizado en tres etapas: Planificación, Ejecución y Documentación. Las actividades relativas a las dos primeras etapas se describen en las siguientes subsecciones y la etapa de documentación se corresponde con la Sección 3.

2.1 Etapa de Planificación

En esta etapa se han llevado a cabo las siguientes actividades:

1. Establecimiento de las preguntas de investigación.
2. Definición de la estrategia de búsqueda.
3. Establecimiento de los criterios de selección de los estudios primarios.
4. Establecimiento de los criterios de evaluación de calidad.
5. Definición de la estrategia de extracción de datos.
6. Selección de los métodos de síntesis.

2.1.1 Preguntas de Investigación

El objetivo principal de este estudio se basa en examinar el estado actual de las auditorías de Green in IT. Para ello, se han establecido las preguntas de investigación que se pueden observar en la Tabla 1.

Tabla 1. Preguntas de investigación.

Preguntas de Investigación	Motivación
Q1. ¿Qué estudios existen sobre auditoria de Green in IT?	Determinar el número de publicaciones actuales y la tendencia a lo largo de los últimos años en relación con el campo de la auditoría de Green in IT.
Q2. ¿Qué metodologías o técnicas se emplean para auditar el Green in IT?	Determinar cuáles son las metodologías y/o técnicas que utilizan los auditores para evaluar el Green in IT y cómo son los informes de auditoría que estos realizan.
Q3. ¿Qué áreas se auditan preferentemente en Green in IT?	Determinar qué áreas son normalmente a las que se presta más atención en las auditorías de Green in IT.
Q4. ¿Qué indicadores se utilizan en las mediciones de auditorías de Green in IT?	Determinar cuáles son los indicadores que suelen utilizar los auditores a la hora de evaluar el Green in IT y cuáles son los valores que estos deben tener.
Q5. ¿Cuáles son los roles relacionados con la auditoría de Green in IT?	Determinar cuáles son los roles implicados en una auditoria de Green in IT.
Q6. ¿Cuáles son los procesos relacionados con la auditoría de Green in IT?	Determinar cuáles son los procesos implicados en una auditoria de Green in IT.

A través de estas preguntas se podrá recopilar y categorizar la información existente acerca de auditorías de Green in IT, con el fin de conocer el estado del arte del área, identificando las carencias existentes para proponer nuevas áreas de investigación.

2.1.2 Estrategia de Búsqueda

Para realizar la búsqueda de información automatizada se va a utilizar la base de datos *Scopus*, en la que se introducirá una cadena de búsqueda dividida en dos partes, que representan, por un lado, el ámbito de la auditoria, y, por otro lado, el Green in IT. En la Tabla 2 se muestra esta cadena de búsqueda, en la cual se han utilizado el booleano OR para unir los términos y sinónimos en cada una de las partes, y el booleano AND para unir las dos partes entre sí.

Tabla 2. Cadena de búsqueda.

Concepto	Términos Alternativos y Sinónimos	
Auditoría	(Audit* OR "Best practices" OR Control* OR Govern* OR Manag* OR Assess* OR Evaluat* OR Measur*)	AND
Green in IT	("Green IT" OR "Green ICT" OR "Green in IT" OR "Green hardware" OR "Green software" OR Greenability)	

Los asteriscos significan que cualquier carácter o caracteres pueden ser incluidos en la palabra en cuestión, de esta manera se consiguen permutaciones en los términos de búsqueda en los que se utiliza (p.ej., el término de búsqueda "Audit" incluye las siguientes palabras: Audit OR Audits OR Auditing OR…).

La búsqueda se realizará aplicando la cadena de búsqueda sobre el título, el *abstract* y las *keywords* de cada artículo.

Por otra parte, se considerará toda aquella información que haya sido publicada en la última década (entre 2005 y 2015, ambos incluidos). Esto es debido a que la idea de Green in IT es relativamente joven y ha sido durante la última década cuando se ha iniciado el desarrollo y los avances en este ámbito. Además, en un campo como el Green in IT que se encuentra en constante crecimiento y renovación, elegir un período de tiempo más extenso puede ser una desventaja, puesto que se encontraría información desactualizada en este contexto. Asimismo, este período se ha validado durante la ejecución del mapeo sistemático al observar que prácticamente todas las publicaciones en el ámbito se encuentran localizadas en los años comprendidos en dicho período.

2.1.3 Criterios de Selección de los Estudios Primarios

La información recopilada en la búsqueda automatizada se va a evaluar teniendo en cuenta el título, *abstract* y *keywords* de cada artículo, con el fin de determinar si dicho artículo va a ser incluido o no. Para ello, por un lado, se incluirán aquellos artículos que cumplan con al menos uno de los siguientes criterios de inclusión:

- I1: artículos en inglés que se refieran a la auditoría en Green in IT.
- I2: artículos completos publicados entre 2005 y 2015 en revistas, conferencias, congresos o talleres de prestigio con revisión por pares.

Por otro lado, los artículos que cumplan con alguno de los siguientes criterios de exclusión, no se tendrán en cuenta:

- E1: tipos de artículos de debate o de opinión, o disponibles solo en forma de resúmenes o presentaciones.
- E2: trabajos duplicados (siempre considerando el artículo más completo y reciente).

- E3: trabajos relacionados con el Green by IT.
- E4: trabajos cuya principal contribución no se relacione con el Green in IT, o en los que el Green in IT se considere de manera superficial.

Las referencias de cada uno de los artículos que se incluyan van a ser evaluadas, es decir, se va a tener en cuenta el efecto de bola de nieve.

2.1.4 Criterios de Evaluación de Calidad

Para medir la calidad de los estudios seleccionados se ha desarrollado un cuestionario con un sistema de puntuación de tres valores (-1, 0 y +1). El cuestionario está formado de las siguientes cuestiones a considerar:

a. El estudio presenta una descripción detallada sobre las características y la aplicación de las auditorías de Green in IT.
b. El estudio contiene guías detalladas sobre cómo realizar auditorías de Green in IT.
c. El estudio valida la idea sobre auditoría de Green in IT que defiende.
d. El estudio expone de manera clara y detallada los resultados obtenidos tras aplicar la idea sobre auditoría de Green in IT que defiende.
e. El estudio ha sido publicado en una revista, conferencia o congreso relevante.
f. El estudio ha sido citado por otros autores.

La suma de la puntuación de cada cuestión conformará la puntuación final de calidad sobre el estudio en cuestión (obteniéndose un valor entre -6 y +6). Estas puntuaciones no se utilizarán para excluir un determinado estudio del mapeo sistemático en el caso de obtener una mala calificación, sino que se utilizará para encontrar estudios más representativos y relevantes, los cuales tendrán más peso en futuras investigaciones.

2.1.5 Estrategia de Extracción de Datos

La estrategia de extracción de datos se basará en una serie de posibles respuestas para cada una de las preguntas de investigación definidas. Gracias a esta estrategia, se asegura la aplicación de los mismos criterios de extracción de datos para todos los estudios seleccionados, facilitando su clasificación. Las preguntas de investigación, así como sus posibles respuestas se pueden observar en la Tabla 3.

Tabla 3. Esquema de clasificación.

Preguntas	Respuestas	
Q1. ¿Qué estudios existen sobre auditoria de Green in IT?	**a.** Mapeo sistemático/revisión de la literatura	
	b. Caso de estudio	**d.** Propuesta
	c. Encuesta	**e.** Otros
Q2. ¿Qué metodologías o técnicas se emplean para auditar el Green in IT?	**a.** COBIT	**d.** Orientada a riesgos
	b. ISO	**e.** Otras
	c. ITIL	**f.** N/A
Q3. ¿Qué áreas se auditan preferentemente en Green in IT?	**a.** Infraestructuras TI	**d.** Gobierno de TI
	b. Aplicaciones software	**e.** Otras
	c. Gestión de TI	**f.** N/A

Preguntas	Respuestas
Q4. ¿Qué indicadores se utilizan en las mediciones de auditorías de Green in IT?	**a.** Potencia (W) **b.** Eficiencia (%, DCIE) **c.** Efectividad (PUE) **d.** Productividad (MFLOPS/W, FLOPS/J…) **e.** N/A
Q5. ¿Cuáles son los roles relacionados con la auditoría de Green in IT?	**a.** Junta directiva **b.** Director ejecutivo (CEO) **c.** Director financiero (CFO) **d.** Dueños del negocio **e.** Comité directivo de sostenibilidad (SSC) **f.** Director de recursos humanos (CHRO) **g.** Director de sistemas de información (CIO) **h.** Director de tecnología (CTO) **i.** Director de sostenibilidad (CSO) **j.** Partes interesadas (*Stakeholders*) **k.** Auditor externo/interno de sostenibilidad **l.** N/A
Q6. ¿Cuáles son los procesos relacionados con la auditoría de Green in IT?	**a.** Evaluar, Orientar y Supervisar (EDM) **b.** Alinear, Planificar y Organizar (APO) **c.** Construir, Adquirir e Implementar (BAI) **d.** Entrega, Servicio y Soporte (DSS) **e.** Supervisar, Evaluar y Valorar (MEA) **f.** N/A

2.1.6 Métodos de Síntesis

En primer lugar, se realizará una síntesis de datos cuantitativa que se basará en:

- Establecimiento y representación a través de tablas y/o gráficos del número y/o porcentaje de los estudios seleccionados clasificados según sus posibles respuestas en cada una de las preguntas de investigación, así como del año de publicación.
- Definición de diagramas de burbuja con el fin mostrar la frecuencia con la que se relacionan las posibles respuestas de cada una de las preguntas de investigación.

Y, en segundo lugar, se llevará a cabo una síntesis de datos cualitativa, basada en:

- Representación a través de tablas y/o gráficos de los estudios seleccionados clasificados según los resultados de las evaluaciones de calidad realizadas.

2.1.7 Calendario del Mapeo Sistemático

El mapeo sistemático tuvo como inicio enero de 2016 y se finalizó en abril de 2016.

2.2 Etapa de Ejecución

La etapa de ejecución, donde se ha aplicado el protocolo de revisión establecido en la etapa anterior, se ha guiado por tres fases principales:

1. En la primera fase, tras aplicar la cadena de búsqueda de la Tabla 2 sobre la base de datos *Scopus*, se obtuvo como resultado un total de 627 documentos, de los cuales, tras aplicar los criterios de selección sobre el *abstract* de cada uno de ellos, se obtuvo un total de 55 estudios potenciales.
2. Durante la segunda fase, se llevó a cabo de nuevo la aplicación de los criterios de selección sobre los 55 estudios potenciales, pero en este caso sobre el estudio en su totalidad. Tras este filtro, se obtuvo un total de 13 estudios primarios.
3. Finalmente, en la última fase se llevó a cabo la evaluación de calidad de cada uno de estos estudios primarios, así como una caracterización de los estudios en tres grupos (tal y como se explica en la Sección 3).

3 Resultados

A continuación se muestran los resultados obtenidos en cada una de las preguntas de investigación, así como en el mapeo sistemático en general.

Sin embargo, antes de nada, es importante destacar una limitación o inconveniente encontrado durante la evaluación de los estudios primarios.

El reducido número de estudios primarios encontrados daba a entrever el poco avance dentro del área de las auditorías de Green in IT. De hecho, de estos estudios únicamente dos ([S01] y [S02]) tratan sobre auditoría de Green in IT. Por ello, con el fin de poder responder a las preguntas de investigación y dar validez al mapeo, se consideró tener en cuenta los once estudios restantes incluidos en los estudios primarios seleccionados. Asimismo, debido a estas circunstancias, se ha creído conveniente caracterizar los 13 estudios primarios en tres grupos para una mejor comprensión:

1. Estudios estrechamente relacionados con auditoría de Green in IT: el cual consta de los estudios [S01] y [S02]; de los cuales, el [S01] trata sobre un análisis del estado del Green IT y recalca la importancia de llevar a cabo auditorías sobre este campo, y el [S02] muestra los resultados de una encuesta realizada a auditores internos de diferentes organizaciones acerca de sus experiencias y opiniones sobre Green IT.
2. Estudios con técnicas de auditoría de Green in IT: en el que se encuentran los estudios [S03] y [S04]. El estudio [S03] trata sobre un modelo para medir y gestionar la madurez del Green IT en una organización. Mientras que el [S04] ofrece, desde la visión del *balanced scorecard*, una serie de iniciativas para evaluar el Green IT. En ambos, las áreas, indicadores y demás características tratadas guardan un símil y estrecha relación con los temas de auditoría.
3. Estudios de Green in IT relacionados con algún otro aspecto de auditoría: formado por el resto de estudios. Estos estudios tratan sobre diversos temas de Green IT, de los que se pueden observar características importantes acerca de esta área, las cuales ayudan a responder las preguntas de investigación.

Por un lado, en el Apéndice A se encuentran todas las referencias a los estudios primarios seleccionados. Y, por otro lado, en el Apéndice B se recopilan los resultados relativos a las respuestas de cada una de las preguntas en relación a cada uno de los estudios primarios seleccionados.

3.1 Pregunta Q1. Estudios sobre Auditorías de Green in IT

Cerca del 40% de los estudios seleccionados son o llevan a cabo un mapeo sistemático o revisión de la literatura, con el fin de ofrecer al lector una base sobre la que desarrollar el resto del estudio ([S06], [S09] y [S10]), o simplemente mostrar el estado del arte del campo en cuestión ([S08] y [S11]).

También, alrededor del 40% de estudios son propuestas ([S03], [S04], [S09], [S10] y [S13]) con las que sus autores pretenden establecer unas guías sobre, por ejemplo, cómo evaluar la madurez del Green in IT en una organización [S03], o sobre un modelo de contingencia para el gobierno de Green in IT [S09].

Por otra parte, el 23% de estudios incluyen algún tipo de encuesta ([S02], [S03] y [S06]), para validar su idea [S03] o mostrar el estado actual en las organizaciones [S02].

Por último, solo el 15% de los estudios ([S09] y [S10]) validan su idea a través de un caso de estudio.

3.2 Pregunta Q2. Metodologías y Técnicas en Auditorías de Green in IT

Solo el 40% de los estudios tienen en cuenta alguna metodología o técnica para gestionar/controlar algún aspecto relacionado con el Green in IT. De estos, la mayoría (el 80%) destacan la importancia de aplicar las normas ISO existentes dentro del Green in IT ([S01], [S05], [S06] y [S13]), en especial el conjunto de normas ISO 14000.

Por otra parte, el estudio [S02] ofrece una metodología orientada a riesgos, mientras que el estudio [S06], además de destacar las normas ISO, centra su visión en demostrar la importancia de ITIL como guía para gestionar el Green in IT.

3.3 Pregunta Q3. Áreas Auditadas en Green in IT

Respecto a qué áreas son las principales dentro del Green in IT y, por lo tanto, las que se deben auditar, prácticamente todos los estudios (el 85%) coinciden en que las dos áreas más importantes son las infraestructuras TI y las aplicaciones software, pues son las que recogen el grueso de las buenas prácticas de Green in IT.

Así mismo, el gobierno de TI también es un área importante en la que más de la mitad de los estudios (54%) hacen especial hincapié, sobre todo en el ámbito de alinear el gobierno tanto corporativo como de TI con el Green in IT.

Por último, la gestión de TI, aunque la menos mencionada con un 38%, es también caracterizada como un área relevante y necesaria para el control del Green in IT.

3.4 Pregunta Q4. Indicadores en las Mediciones en Auditorías de Green in IT

El 69% de los estudios respaldan que el indicador principal en las mediciones de Green in IT es la potencia, es decir, el consumo en vatios (W) de las TI.

Otros estudios como son el [S04], [S11] y [S13], defienden también a la eficiencia de las TI como un indicador principal. Por otro lado, los estudios [S04] y [S13] también defienden el papel de la efectividad como uno de los indicadores principales e importantes en el Green in IT.

Por su parte, la productividad únicamente se cita por el estudio [S11].

3.5 Pregunta Q5. Roles relacionados con las Auditorías de Green in IT

Dentro de los roles, los que más destacan con un respaldo del 54% de los estudios son los de CIO y CTO, los principales encargados del Green in IT dentro de las organizaciones.

Tras estos, el 38% de los estudios también destacan la importancia dentro del Green in IT del CEO y del director/responsable de sostenibilidad (CSO, rol del cual pocas empresas cuentan según se ha podido observar).

Por su parte, la junta directiva y el CFO son roles a tener en cuenta según el 23% de los estudios, sobre todo en temas de gestión financiera y alineación con el negocio.

Asimismo, el comité directivo de sostenibilidad (SSC), aunque prácticamente ausente en la mayoría de organizaciones, el 15% de los estudios destacan la importancia de la existencia de un comité de tales características en este campo.

Por último, el estudio [S02] también destaca el papel de los auditores, especialmente internos, en el ámbito que nos atañe. Destacar que este es el único estudio que da una visión puramente práctica de la auditoría.

3.6 Pregunta Q6. Procesos relacionados con las Auditorías de Green in IT

Los procesos más importantes dentro del Green in IT según el 85% de los estudios y, por tanto, los que necesariamente deber ser controlados a través de auditorías, son aquellos que están relacionados con el ámbito de construir, adquirir e implementar (BAI) técnicas y/o buenas prácticas de Green in IT.

Aquellos procesos que tienen relación con la gestión del Green in IT, como son los relacionados con EDM y MEA, también son muy importantes de auditar, según corrobora el 77% de los estudios. Cualquier fallo en estos procesos que se encargan de supervisar, evaluar, valorar, etc., provoca el mal funcionamiento de las prácticas de Green in IT implementadas, por ello son vitales de controlar.

No menos importantes son los procesos relacionados con alinear, planificar y organizar (APO), que cuentan con el respaldo del 69% de los estudios. Estos procesos, eminentemente relacionados con el gobierno, son sumamente importantes de controlar debido a que si la cabeza de una organización no se encuentra alineada y comprometida con el Green in IT, es difícil que esta idea y buenas prácticas implementadas consigan asentarse con éxito dentro de dicha organización.

Por último, los procesos relacionados con la entrega, servicio y soporte (DSS), es decir, los procesos más relacionados de cara a los clientes, solo son nombrados en los estudios [S04] y [S13], como importantes dentro del Green in IT.

3.7 Resultados del Mapeo

Tras analizar cada una de las preguntas en cuestión, como resultados generales del mapeo se puede determinar lo siguiente:

- La mayoría de estudios actuales son propuestas y/o mapeos o revisiones de la literatura en general sobre Green IT y algunas incluyen Green in IT, lo que define y demuestra la novedad del campo en cuestión.
- Las normas ISO (en especial la serie 14000) tienen que tenerse en cuenta cuando se lleve a cabo la adopción o implementación del Green in IT en las organizaciones.
- Las infraestructuras TI y las aplicaciones software son las áreas principales sobre las que se llevan a cabo las buenas prácticas y/o técnicas de Green in IT.
- El Green in IT se traduce principalmente en reducir el consumo en vatios (W) de las TI, por lo que la potencia es el principal indicador a tener en cuenta.
- El CIO y el CTO son los principales responsables dentro del Green in IT en las organizaciones.
- Los procesos relacionados con la construcción, adquisición e implementación (BAI) de las técnicas y/o buenas prácticas de Green in IT, son los más relevantes y sobre los que se desarrollan el mayor número de actividades (seguidos muy de cerca por los procesos de EDM y MEA).

Por otra parte, debido al escaso número de estudios encontrados y los problemas que se han tenido para responder a las preguntas de investigación (por la poca relación con el tema), no tiene sentido realizar gráficos característicos de este tipo de mapeos, como los diagramas de burbuja, ya que no arrojarían ninguna información de interés o esclarecedora.

Sin embargo, hemos considerado que la pregunta Q6 relacionada con los procesos implicados en la auditoría de Green in IT es importante de destacar. Además, las respuestas a esta pregunta están directamente relacionadas con los procesos del marco COBIT de auditoría, el cual puede servir de base para el desarrollo de un marco de auditoría de Green in IT; por lo que el interés en conocer qué procesos son los más importantes en el tema en cuestión es mayor. Así pues, como se ha visto en los resultados de esta pregunta, los procesos de BAI, EDM y MEA toman una relevancia especialmente importante en este campo del Green in IT, ya que aparecen mediante algún tipo de relación en más del 75% de los estudios analizados.

4 Discusión

4.1 Observaciones Principales

El objetivo del presente mapeo sistemático es el de conocer el estado actual del campo de la auditoría de Green in IT, con el fin de determinar las características más importantes para desarrollar un marco de auditoría de Green in IT. Así pues, tras analizar los resultados se pueden deducir las siguientes observaciones:

- Prácticamente nula existencia de investigaciones relacionadas con la auditoría de Green in IT. El reducido número de estudios existentes demuestra que es un campo novedoso y vital de desarrollar (como destaca [S01]).
- Creciente importancia del Green IT. La mayoría de los estudios destacan la gran importancia de este campo, cuyo interés cada día es mayor y la relevancia que está tomando en la sociedad lo está convirtiendo en un área indispensable tanto para el futuro de las organizaciones como el de la humanidad.

4.2 Limitaciones del Mapeo Sistemático

La principal limitación del presente mapeo sistemático se basa en el establecimiento de una cadena de búsqueda muy específica en el tema de la auditoría de Green in IT. Durante la realización del mapeo hemos observado que, por ejemplo, en relación al término "Green in IT" se podrían haber utilizado términos más generales (como *sustainability*), pero dichos términos devolverían un elevadísimo número de resultados y muchos de ellos poco o nada relacionados con el Green IT o con las TI en general. Por ello, y teniendo en mente el objetivo primordial del estudio, se decidió no realizar una búsqueda tan amplia en esta primera toma de contacto con este tema, sino llevarla a cabo más adelante una vez se hayan establecido las bases que se pretenden alcanzar.

4.3 Trascendencia para la Investigación y la Práctica

Las observaciones del presente mapeo sistemático tienen una gran trascendencia para aquellos investigadores que estén planeando investigar sobre Green IT, y más específicamente en el tema de auditorías de Green in IT; y, por supuesto, para auditores, directivos de informática, u otros responsables en sus organizaciones del Green in IT.

En primer lugar, para los investigadores es un área muy interesante, ya que, como se ha visto, se trata de un campo novedoso, en el que prácticamente no hay nada hecho.

En segundo lugar, los auditores gracias al desarrollo de este campo podrán desarrollar nuevos modelos de auditoría que les permitan ampliar su radio de acción hacia este nuevo campo del Green IT. De esta forma se conseguirán realizar auditorías más específicas y completas, ayudando a consolidar las buenas prácticas de Green IT.

Y, por último, las organizaciones se verán en gran medida beneficiadas por el avance de este campo. Hasta el momento algunas organización han desarrollado e implementado según sus propios criterios aquellas medidas o prácticas de Green IT que han considerado oportunas, sin llevar a cabo ningún tipo de auditoría interna o externa

validada. Asimismo, más organizaciones podrán unirse a estas buenas prácticas pues no tendrán que inventar sus propias técnicas, sino que podrán acceder al conocimiento que existe sobre el área y adaptarlo a su situación de una manera sencilla.

5 Conclusiones y Trabajo Futuro

En los últimos años el Green IT se ha convertido en una de las áreas más importantes y relevantes, y en un futuro cercano su aplicación será indispensable [5] [6]. Aunque se trata de una idea relativamente joven y su implementación y técnicas aún se encuentran en fases muy tempranas, cada vez son más las organizaciones que deciden seguir este tipo de técnicas.

Los resultados obtenidos en el mapeo sistemático presentado en este artículo demuestran la novedad de este campo y la necesidad de elaborar un marco de auditoría de Green in IT que permita controlar las prácticas adoptadas por las organizaciones al respecto y, también, establecer unas guías de buenas prácticas validadas empíricamente.

Por ello, es importantísimo seguir en el desarrollo de esta idea, y, respecto al trabajo futuro, ya estamos trabajando en:

- Desarrollo de un nuevo mapeo sistemático en el que se aplique una cadena de búsqueda más amplia y se incluyan más motores de búsqueda. En este nuevo mapeo se pretende paliar la limitación explicada en la sección anterior y ampliar el objeto de estudio al Green IT en general.
- Elaboración de un marco de auditoría de Green in IT, que servirá de base para un posterior marco de auditoría de Green IT.

Agradecimientos. Esta investigación es parte del proyecto GINSENG (TIN2015-70259-C2-1-R) financiado por el Ministerio de Economía y Competitividad de España y por el fondo FEDER (Fondo Europeo de Desarrollo Regional); y GLOBALIA (PEII-2014-038-P), Consejería de Educación y Ciencia, Junta de Comunidades de Castilla-La Mancha.

Referencias

1. Budgen, D., Turner, M., Brereton, P., Kitchenham, B.: Using mapping studies in software engineering. In: Proceedings of PPIG 2008, pp. 195-204. Lancaster University (2008)
2. Calero, C., Piattini, M.: Green in Software Engineering. Springer International Publishing AG, Cham, ZG, Switzerland (2015)
3. Du, W., Pan, S. L., Zuo, M.: How to Balance Sustainability and Profitability in Technology Organizations: An Ambidextrous Perspective. In: IEEE Transactions on Engineering Management, Vol. 60, Issue 2, pp. 366-385 (2013)
4. Erdélyi, K.: Special factors of development of green software supporting eco sustainability. In: IEEE 11th International Symposium on Intelligent Systems and Informatics (SISY), pp. 337-340. Subotica, Serbia (2013)
5. Esteve Zarazaga, F. J.: Grupo "Green ICT" de CEPIS. In: Novática, Num. 234, pp. 13 (2015)

6. Esteve Zarazaga, F. J.: Las "TIC verdes" en el Horizonte 2025. In: Novática, Num. 234, pp. 80-84 (2015)
7. Genero Bocco, M., Cruz-Lemus, J. A., Piattini Velthuis, M. G.: Métodos de investigación en ingeniería del software. Ra-Ma Editorial, Madrid, Spain (2014) 199-246
8. Kitchenham, B.: Guidelines for Performing Systematic Literature Reviews in Software Engineering, Version 2.3. EBSE Technical Report, Keele University, UK (2007)
9. Petersen, K., Feldt, R., Shahid, M., Mattsson, M.: Systematic mapping studies in software engineering. In: 12th International Conference on Evaluation and Assessment in Software Engineering (EASE). Department of Informatics, University of Bari, Italy (2008)
10. Unhelkar, B.: Green IT Strategies and Applications: Using Environmental Intelligence. CRC Press, Boca Raton, FL, USA (2011)
11. COBIT 5 - An ISACA Framework, http://www.isaca.org/cobit

Apéndice A: Estudios Primarios Seleccionados

S01. Gabriel, C.: Why it's not naive to be green. In: Business Information Review, Vol. 25, pp. 230-237 (2008)
S02. Gray, G. L., No, W. G., Miller, D. W.: Internal Auditors' Experiences and Opinions Regarding Green IT: Assessing the Gap in Normative and Positive Perspectives. In: Journal of Information Systems, Vol. 28, pp. 75-109 (2013)
S03. Park, S.-H., Eo, J., Lee, J. J.: Assessing and Managing an Organization's Green IT Maturity. In: MIS Quarterly Executive, Vol. 11 (2012)
S04. Jain, R., Benbunan-Fich, R., Mohan, K.: Assessing green IT initiatives using the balanced scorecard. In: IT Professional Magazine, Vol. 13, pp. 26-32 (2011)
S05. Agarwal, S., Nath, A.: Green computing-a new horizon of energy efficiency and electronic waste minimization: a global perspective. In: 2011 International Conference on Communication Systems and Network Technologies (CSNT). Jammu, India (2011)
S06. Cater-Steel, A., Tan, W.-G.: The role of IT service management in Green IT. In: Australasian Journal of Information Systems, Vol. 17 (2011)
S07. Chou, D. C., Chou, A. Y.: Awareness of Green IT and its value model. In: Computer Standards & Interfaces, Vol. 34, pp. 447-451 (2012)
S08. Loeser, F.: Green IT and Green IS: Definition of constructs and overview of current practices. In: 19th Americas Conference on Information Systems (AMCIS). Chicago, IL, USA (2013)
S09. Schmidt, N.-H., Kolbe, L. M.: Towards a contingency model for green IT governance. In: European Conference on Information Systems (ECIS). Helsinki, Finland (2011)
S10. Opitz, N., Krüp, H., Kolbe, L. M.: How to Govern your Green IT? - Validating a Contingency Theory Based Governance Model. In: 19th Pacific Asia Conference on Information Systems (PACIS). Chengdu, China (2014)
S11. Ardito, L., Morisio, M.: Green IT - Available data and guidelines for reducing energy consumption in IT systems. In: Sustainable Computing: Informatics and Systems, Vol. 4, pp. 24-32 (2014)
S12. Murugesan, S.: Harnessing green IT: Principles and practices. In: IT professional, Vol. 10, pp. 24-33 (2008)
S13. Wati, Y., Koo, C.: An Introduction to the Green IT balanced scorecard as a strategic IT management system. In: 2011 44th Hawaii International Conference on System Sciences (HICSS). Kauai, HI, USA (2011)

Apéndice B: Resultados de los Estudios Primarios

Tabla 4. Resultados de los estudios primarios.

ID	Q1					Q2						Q3						Q4					Q5						Q6					
	a	b	c	d	e	a	b	c	d	e	f	a	b	c	d	e	f	a	b	c	d	e	a	b	c	d	e	f	g	h	i	j	k	l	a	b	c	d	e	f
S01					X	X						X	X					X										X	X					X	X			X
S02			X						X			X	X					X					X	X	X			X	X	X		X		X	X			X
S03		X	X									X	X	X	X	X						X										X		X	X	X		X
S04			X									X	X	X				X	X	X			X	X	X				X	X	X			X	X	X	X	X
S05				X		X		X					X	X				X														X			X			
S06	X		X		X	X	X						X	X	X	X		X										X	X					X	X	X	X	X
S07				X								X	X	X	X	X		X														X	X	X	X	X		X
S08	X											X	X	X	X	X		X														X	X	X	X	X		X
S09	X	X		X		X						X			X							X		X			X	X	X						X			
S10	X	X		X								X			X							X		X			X	X	X						X			
S11	X											X	X	X				X	X		X											X	X		X		X	
S12					X							X	X	X	X	X		X														X	X	X	X		X	
S13			X			X							X	X					X	X			X	X	X			X	X	X					X	X	X	X	X

Herramienta de Soporte a la Evaluación y Mejora de la Gestión de Planes de Emergencia

Ana-Gabriela Núñez, Mª Carmen Penadés, José H. Canós
ISSI - DSIC
Universitat Politècnica de València
Camino de Vera s/n, 46022 Valencia
{anunez | mpenades | jhcanos}@dsic.upv.es

Resumen. La gestión de planes de emergencia es un tema que compete a todas las organizaciones y a la comunidad en general. La falta de propuestas para evaluar un plan de emergencia, más allá de una simple auditoría, hace que los planificadores construyan planes de emergencia basándose en su propia formación y experiencia. A pesar de las regulaciones legales existentes sobre formato y contenido de los mismos, pensamos que la definición de un marco de referencia para evaluar su gestión, aumentará la capacidad de los planificadores de construir cada vez mejores planes. QuEP define un modelo de evaluación y mejora basado en niveles de madurez, respecto a la gestión que una organización hace de su plan de emergencia; cada nivel identifica principios, prácticas, cuestiones y técnicas. Una vez realizada la evaluación de la organización, se le sugieren las técnicas a seguir para mejorar el ciclo de vida de su plan de emergencia, aumentando así la calidad del mismo, y por tanto, mejorando la gestión de emergencias en la organización. En este artículo se presenta el desarrollo iterativo de una herramienta web de soporte al marco QuEP para la evaluación de la gestión de planes de emergencia, generando la estrategia de mejora a seguir como un conjunto de buenas prácticas organizadas por niveles y por actores.

Palabras clave: Modelo de evaluación y mejora, Gestión de Planes de Emergencia, Marco QuEP, Herramienta de soporte.

1 Introducción

En los últimos años, la Gestión de Emergencias ha comenzado a tomar relevancia dentro de la investigación científica y práctica, dada la creciente ocurrencia tanto de eventos naturales, comoprovocados por el hombre. La respuesta a situaciones de emergencia está dirigida por el plan de emergencia, que es el principal activo generado en la etapa de planificación. Es un documento que recoge las acciones realizadas en respuesta a incidentes potenciales junto con otra información, tal como descripciones de la infraestructura y riesgos. También se utiliza en las organizaciones como herramienta de formación, tanto de empleados y usuarios como de los equipos de respuesta encargados de atender las potenciales situaciones de riesgo.

Pese a la importancia de los planes de emergencia, poco o nada se ha hecho para definir claramente qué es un buen plan, ni cómo aplicar los planes en las

organizaciones. En ausencia de cualquier tipo de estándar de plan de emergencia, diferentes países han publicado guías o leyes que definen su contenido mínimo, y que las organizaciones deben seguir. Entre ellas se encuentran, por ejemplo, la "Ley de Autoprotección" (NBA) [1] en España, la "Comprehensive Preparedness Guide" (CPG) 101 [2] en los EE.UU o la "Comprehensive Preparedness Guide" [3] en Reino Unido, "Guía para elaborar un plan de emergencia y evacuación en edificios" (SIGWEB) [4] en Chile, "Emergency Management Planning Guide 2010–2011" [5] en Canadá y "Emergency Response Management" [6] en Japón. Todas ellas se centran en los aspectos más descriptivos de los planes de emergencia, y algunas como la NBA también en aspectos básicos de gestión administrativa de los mismos. Sin embargo, son prácticamente inexistentes las propuestas de marcos de referencia que permitan la evaluación continua de los planes de emergencia en organizaciones.

QuEP (**Qu**ality of **E**mergency **P**lans Management, [7]) es un marco para la evaluación y mejora de la gestión de planes de emergencia creado para llenar el vacío metodológico en el desarrollo y explotación de los mismos. Inspirado en la Gestión de Calidad Total (GCT) [8], comprende una serie de principios, prácticas y técnicas -siguiendo el modelo de Dean y Bowen [9]- para la mejora en los procesos de gestión de los planes de emergencia, desde su elaboración a su explotación, pasando por el mantenimiento y mejora.

En este artículo presentamos el desarrollo de una herramienta de soporte al marco QuEP, siguiendo un modelo de proceso iterativo, basado en SCRUM. La herramienta, a la que denominamos QuEP-T, permite a los participantes acceder al sitio web con el fin de evaluar la organización y obtener la estrategia de mejora a seguir, como un conjunto de buenas prácticas organizadas por niveles y por actores. Se muestra la puesta en marcha del proyecto y la versión obtenida tras la primera iteración.

El resto del artículo se encuentra estructurado de la siguiente forma. En la Sección 2 se resumen las principales características del marco QuEP. En la Sección 3, se muestra el proceso de desarrollo iterativo y la visión general del proyecto QuEP-T. En la Sección 4 se resume la primera iteración, mostrando el diseño y la arquitectura de la versión 1.0 obtenida. La planificación del resto de iteraciones se resume en la Sección 5, y finalmente, la Sección 6 contiene las conclusiones y trabajos futuros.

2 QuEP: Un modelo de evaluación y mejora para la Gestión de Planes de Emergencia.

Existen diferentes modelos para la GCT entre los que destacamos EFQM [10], Deming [11], ISO 9000:2000 [12], y el Iberoamericano [13]. Estos modelos permiten la evaluación de los procesos de las organizaciones a través de valores cuantificables para incorporar mejoras y comprobar el funcionamiento y rendimiento organizativo. Dentro del ámbito de las Emergencias, Berke y Godschalk [14] señalan la importancia de la evaluación del plan de emergencia, y en esta línea, el trabajo de Meyerson [15], propone una herramienta basada en una lista de verificación que evalúa el contenido de los planes de respuesta en el contexto de las autoridades locales y los planes de mitigación, basada en los principios establecidos por Berke [16]. Pero, no hay propuestas generales o marcos de referencia que aborden la gestión de la planificación de una forma integral,

y permitan la mejora de las actividades involucradas en su gestión, con miras a obtener planes de emergencia de mayor calidad en la organización.

El marco QuEP ([7], [17]) pretende evaluar la madurez que puede alcanzar una organización en la gestión de su plan de emergencia. Se han establecido una serie de principios basados en la GCT [18], pero adaptados y contextualizados al dominio de la gestión de planes de emergencia. Estos principios nos han permitido definir una serie de prácticas o actividades asociadas a los mismos, así como las técnicas propuestas para su correcta realización, en función del rol que desempeña el participante. El objetivo de QuEP no es solamente medir el nivel de madurez en el que se encuentra una organización, sino que a la vez proporcionarle cuáles son las mejores prácticas que le permitirán mejorar su proceso de planificación del plan de emergencia, aumentando así su calidad.

Niveles de Madurez. En QuEP se han definido diez niveles de madurez. En la Figura 1, se presentan dichos niveles agrupados en tres bloques principales: técnico, humano y estratégico, siguiendo la aproximación de Camison [19]. El nivel 1 es el más bajo, y en él se evalúa si la organización es capaz de generar un plan de emergencia como un documento de acuerdo a las normativas o leyes vigentes, pero sin ningún tipo de proceso estructurado de generación del mismo. En el resto de niveles del bloque técnico, del nivel 2 al 4, se evalúa si la organización tiene incorporado un proceso de planificación específico y repetible que garantice la calidad del plan de emergencia, si la organización utiliza un sistema de soporte a la planificación, y finalmente si optimiza sus procesos de planificación, respectivamente.

A partir del nivel 5, se introduce de forma explícita la perspectiva humana, mediante la que se evalúa la participación de todos los involucrados en la gestión de los planes de emergencia (la organización, los planificadores, los trabajadores, los equipos de respuesta y los ciudadanos), la optimización de costes en entrenamientos y simulacros y si se utiliza re-ingeniería. Aspectos más estratégicos, como son el apoyo de la dirección y el liderazgo, o reingeniería global del proceso, considerando todos los aspectos para que aumente la percepción de seguridad, se evalúan en los niveles 8 y 9. Por último, el nivel 10, es el más alto y representa la búsqueda continua de la excelencia.

Modelo de evaluación y mejora. Los niveles de madurez son soportados por un modelo de evaluación y mejora en el proceso de planificación en una organización [17]. Este modelo se define en base a principios, prácticas, técnicas, preguntas y los participantes o involucrados. En la figura 2.a, se presentan los diferentes componentes que conforman dicho modelo. Se han definido nueve principios, los cuales tienen el objetivo de guiar a la organización en la gestión de la calidad respecto a los planes de emergencia y su proceso de planificación, incluyendo las acciones de todos sus participantes. Los principios han sido implantados mediante actividades o prácticas que son realizadas por todos los participantes de la organización en la gestión del plan de emergencia. Para nuestro modelo se han identificado 26 prácticas asociadas a los diferentes principios (véase la figura 2.b).

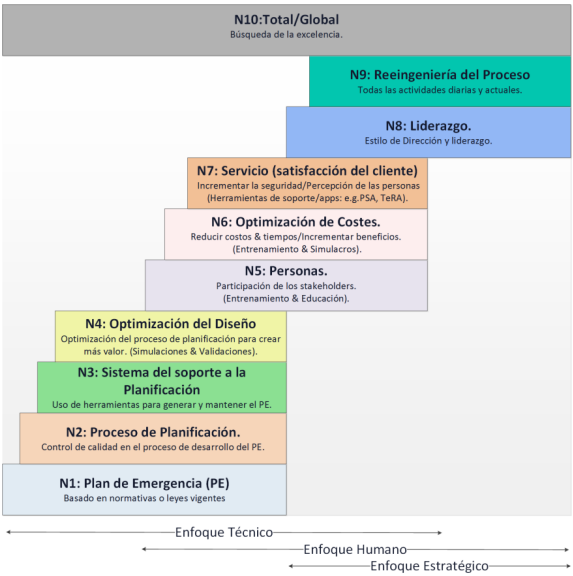


Figura 1. Niveles de Madurez

La realización de las prácticas por parte de la organización se evalúa mediante un conjunto de preguntas asociadas a cada una de ellas. Por otra parte, las prácticas cuentan con una serie de técnicas relacionadas (véase la figura 2.c), que proporcionan a la organización el conocimiento de lo que debe realizar para aplicar las mejores prácticas en la gestión de su plan de emergencia. Además, el modelo QuEP tiene en cuenta los participantes involucrados en las diferentes actividades de la gestión de planes de emergencia y sus responsabilidades o roles, siendo cinco los identificados y que se presentan listados en la figura 2.d.

Actualmente el marco QuEP incluye 176 preguntas repartidas entre los 9 principios y las 26 prácticas [17]. Para su validación por expertos en planificación y gestión de riesgos, se ha definido un proceso Delphi [20] mediante el cual se espera recolectar comentarios y sugerencias de mejora del mismo.

Con el fin de facilitar y automatizar al máximo la evaluación de una organización, así como generar los informes de mejora correspondientes para dar a conocer cuáles son las mejores prácticas que le permitirán mejorar su calidad antes, durante y después del proceso de planificación, se ha diseñado una herramienta de acompañamiento del marco QuEP a la que hemos denominado QuEP-T, y cuyo desarrollo se describe en las siguientes secciones.

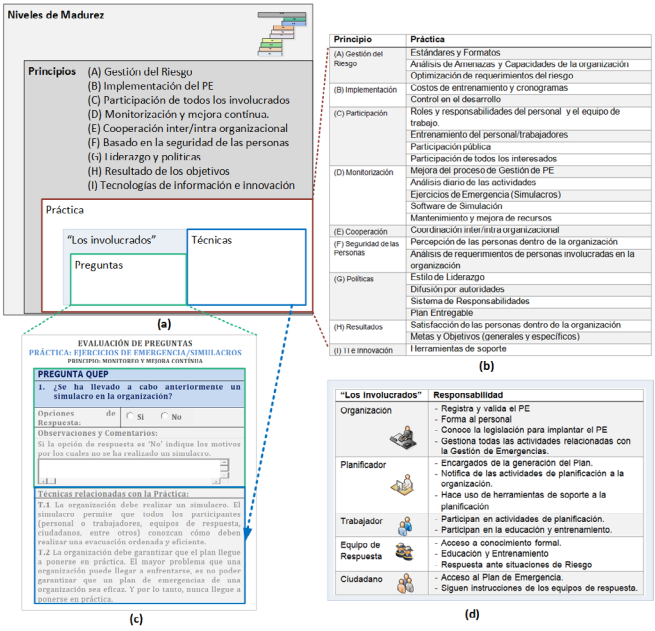


Figura 2. Modelo de Evaluación y Mejora (Marco QuEP)

3 QuEP - T: Proceso de Desarrollo

El desarrollo de QuEP-T se está realizando siguiendo la metodología ágil de gestión de proyectos SCRUM [21]. Su elección ha venido motivada principalmente por la necesidad de hacer un desarrollo iterativo y obtener versiones funcionales en breves periodos de tiempo, para que se puedan ir aplicando a casos reales, y obtener realimentación, que nos permita mejorar tanto el marco QuEP como la herramienta.

La figura 3 resume el proceso SCRUM que estamos siguiendo en el desarrollo de QuEP-T. El rol de *product owner* es desempeñado por un miembro del equipo ISSI-DSIC, pero antes de la toma de cualquier decisión, se requiere de la puesta en común tanto de expertos en planificación de emergencias y gestión de riesgos, como del equipo ISSI-DSIC como expertos en el marco QuEP. Tras valorar todo el conocimiento e información aportada por cada parte, el *product owner* toma las decisiones. Una vez identificado el *product backlog* con el conjunto de unidades de trabajo a realizar, y a las que denominaremos con la abreviatura WUs, el equipo de desarrollo ISSI inicia la

iteración, con duración de tres semanas, para generar la primera versión de QuEP-T. Tal como define SCRUM, nuevas versiones pueden ser generadas en iteraciones posteriores. A continuación, se muestra la puesta en marcha del proyecto, presentando una visión general de QuEP-T v1.0.

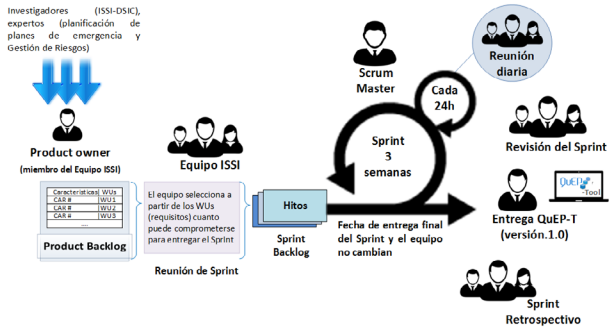


Figura. 3 SCRUM como metodología de desarrollo de QuEP-T

3.1 Puesta en marcha del proyecto QuEP-T

Para tener una visión general de la herramienta, se ha realizado un modelo de dominio y se han identificado los distintos tipos de usuarios que pueden interactuar con la misma. También se han identificado los requisitos funcionales, representados como características, y posteriormente se han refinado en unidades de trabajo (WUs) para determinar el *product backlog*.

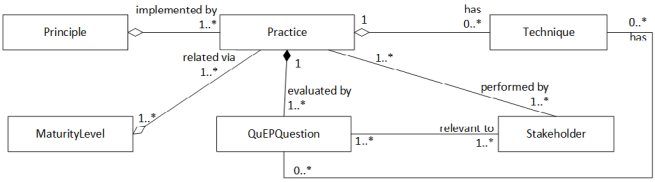


Figura 4. Modelo de dominio QuEP.

Modelo de dominio. Los principales elementos del marco QuEP se han representado como clases, y sus dependencias como diferentes tipos de relaciones (asociaciones, agregaciones y composiciones). La figura 4 resume el marco QuEP como un diagrama

de clases UML. Los principios (*clase Principle*) se implementan mediante conjuntos de prácticas (*clase Practice*), que a su vez están asociados a niveles específicos de madurez (*clase MaturityLevel*) y son realizados por los participantes (*clase Stakeholder*). Las técnicas (*clase Technique*) forman parte de las prácticas y se encuentran asociadas con preguntas (*clase QuEPQuestion*).

Tipos de usuarios. Se han identificado siete tipos de usuario: el *visitante* que consulta información general del marco QuEP; el *administrador*, que como usuario registrado en la herramienta es el encargado de realizar la gestión de usuarios, la gestión de la información del marco y de los cuestionarios de evaluación; la *organización*, el *planificador*, el *trabajador*, el *equipo de respuesta* y el *ciudadano*, los cuales, en función de su rol, participarán como usuarios registrados en la evaluación de la gestión de planes de emergencia en su organización.

Características. La Tabla 1 muestra los requisitos funcionales que se han identificado, a partir del marco QuEP y del modelo de dominio. Son 8 funcionalidades de alto nivel de granularidad que aportan valor a la herramienta y a las que denominaremos características. Están relacionadas tanto con la configuración del marco QuEP, como con la gestión de la organización, sus usuarios y roles, los cuestionarios de evaluación y las propuestas de mejora.

Product Backlog. De acuerdo con la visión general de la herramienta, el *product owner* detalla las WUs que formarán parte del *product backlog* inicial y que se muestran en la Tabla 2. Estas WUs están ordenadas por prioridad, siendo la más prioritaria, la WU1 (*Desplegar cuestionarios a usuarios por rol*) y la menos prioritaria, la WU26 (*Borrado de organizaciones*). En la tabla se muestra, para cada WU, a qué característica está asociada y cuál es su estimación. Dicha estimación corresponde a horas de diseño/programación de la WU. Este *product backlog* puede ser modificado a lo largo del proceso de desarrollo con la inclusión de nuevas funcionalidades para la herramienta, posibles mejoras o corrección de fallos; en cualquier caso, nuevas WUs son incorporadas al *product backlog*, que debe ser nuevamente reorganizado por prioridad antes de iniciarse una nueva iteración.

Tabla 1. Características de QuEP-T

Identificador	Descripción
CAR 1	Configuración del Marco
CAR 2	Gestión de Usuarios y Roles
CAR 3	Gestión de la Organización a evaluar
CAR 4	Configuración de Preguntas
CAR 5	Gestión de Cuestionarios dinámicos
CAR 6	Gestión de la Evaluación de la Organización
CAR 7	Gestión de la Mejora de la Organización
CAR 8	Gestión de Resultados

Tabla 2. Producto Backlog de QuEP-T

Característica	ID	Nombre	Estimación
CAR 5	WU1	Desplegar cuestionarios a usuarios por rol	32
CAR 8	WU2	Visualizar resultados obtenidos de la evaluación para una organización	16
CAR 2	WU3	Control de acceso usuarios	8
CAR 4	WU4	Configurar los tipos de preguntas.	4
CAR 4	WU5	Modificación de tipos de preguntas	4
CAR 4	WU6	Crear preguntas	4
CAR 4	WU7	Modificar preguntas	4
CAR 4	WU8	Configurar opciones de respuesta para cada pregunta	16
CAR 4	WU9	Asignar roles, técnicas y características de resiliencia a las preguntas	8
CAR 5	WU10	Creación de los cuestionarios dinámicos	24
CAR 5	WU11	Modificación de los cuestionarios dinámicos	16
CAR 2	WU12	Asignar usuarios a la organización	8
CAR 6	WU13	Evaluar cuestionarios disponibles para un usuario según el rol	8
CAR 6	WU14	Notificar evaluación de cuestionarios contestados	8
CAR 7	WU15	Visualizar las mejores prácticas (técnicas y características resilientes)	8
CAR 7	WU16	Notificar las técnicas que debe seguir la organización para mejorar su gestión.	8
CAR 2	WU17	Creación de roles y usuarios.	8
CAR 2	WU18	Modificación de usuarios y roles	8
CAR 2	WU19	Borrado de usuarios y roles	4
CAR 1	WU20	Crear componentes (niveles de madurez, principios, prácticas, técnicas y características resilientes) del Marco QuEP	24
CAR 1	WU21	Modificar componentes (niveles de madurez, principios, prácticas, técnicas y características resilientes) del Marco QuEP	16
CAR 3	WU22	Creación de organizaciones	8
CAR 3	WU23	Modificación de organizaciones	8
CAR 3	WU24	Borrado de organizaciones	4

4 QuEP- T: Primera iteración

La planificación de la primera iteración del proyecto QuEP-T se muestra en la Tabla 3. Podemos observar que se han seleccionado las tres WUs de mayor prioridad del *product backlog*, y que la duración de la iteración será de tres semanas (considerando un total de 120 horas de diseño/implementación, acorde con las estimaciones realizadas). Además se han detallado las tareas que el propio equipo de desarrollo ha incluido como parte de la iteración, para lograr el objetivo, la v1.0 de QuEP-T.

Tabla 3. Planificación de la 1ª iteración del proyecto QuEP-T

Iteración	ID	Nombre WUs/tarea	Estimación
Iteración 1 (3 semanas)	WU1	Desplegar cuestionarios a usuarios por rol	32
	WU2	Visualizar resultados obtenidos de la evaluación para una organización	16
	WU3	Control de acceso usuarios	8
	Tarea1	*Investigación de la tecnología y herramientas necesarias para creación del proyecto QuEP-T.*	8
	Tarea 2	*Analizar y Diseñar el Modelo de datos*	16
	Tarea 3	*Creación y despliegue el proyecto basado en las tecnologías seleccionadas.*	8
	Tarea 4	*Creación de la Base de Datos e ingreso de datos*	16
	Tarea 6	*Crear e implementar patrón MVC para el proyecto QuEP-T*	8
	Tarea 7	*Creación de plantillas para la vista (1era-versión)*	8

ID: **Wu1**	Nombre: **Desplegar cuestionarios a usuarios por rol**	Prioridad: **Muy Alta**
Usuario: Organización, ciudadanos, trabajadores, planificadores y equipos de respuesta.	Riesgo: **Alto**	Estimación: **32 h**
Descripción: Luego que el usuario se identifica, se le presenta desplegado el cuestionario a evaluar con preguntas de una práctica (seleccionada previamente). Las preguntas estarán disponibles según el rol que el usuario tenga. Una vez completadas las respuestas a las preguntas formuladas, se almacena la información y el sistema confirma la correcta recepción de las respuestas.		

Figura 5. Plantilla de descripción de la WU1 "Desplegar cuestionarios a usuarios por rol"

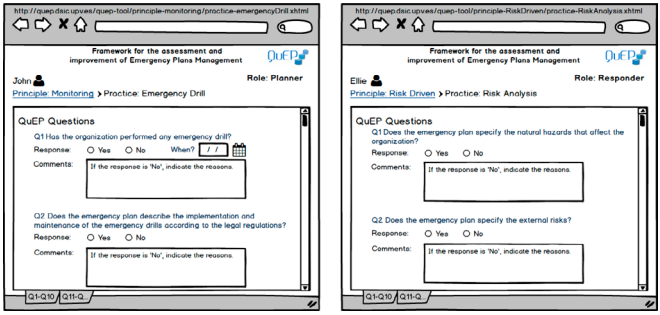


Figura 6. Mockups diseñados para la WU1 "Desplegar cuestionarios a usuarios por rol"

Por cuestiones de espacio, en el resto de la sección sólo se muestran los artefactos más relevantes generados para la WU1 "Desplegar cuestionarios a usuarios por rol", como

son: la plantilla de descripción (ver Figura 5), el diseño preliminar mediante mockups (ver Figura 6) y un esbozo de las pruebas de aceptación definidas (ver Figura 7). Para la WU1, existirá una prueba de aceptación como la que muestra la Figura 7, para cada uno de los roles por cada práctica QuEP. En total se han definido 130 pruebas de aceptación para la WU1.

Nombre Prueba:	**Desplegar correctamente los cuestionarios a usuarios por roles**
Condiciones de ejecución:	• Exista el usuario "John" con el rol "Planner". • Existan preguntas configuradas
Pasos:	• El usuario "John" ingrese y se autentifique en QuEP-T. • Seleccione del menú la práctica "Emergency Drill"
Resultado esperado:	• Se visualicen para la práctica "Emergency Drill" (ED: Ejercicios de Emergencia/Simulacros) las 24 quep questions (Qx) definidas previamente. (ED.Q1, ED.Q2, ED.Q3,...ED.QN).

Figura 7. Prueba de aceptación de la WU1 "Desplegar cuestionarios a usuarios por rol"

4.1 Modelo de diseño de QuEP-T

El modelo de datos diseñado e implementado para la primera iteración contiene todas las entidades relevantes del modelo QuEP (niveles de madurez, principios, prácticas, técnicas, preguntas e involucrados), como se puede ver en la Figura 8.

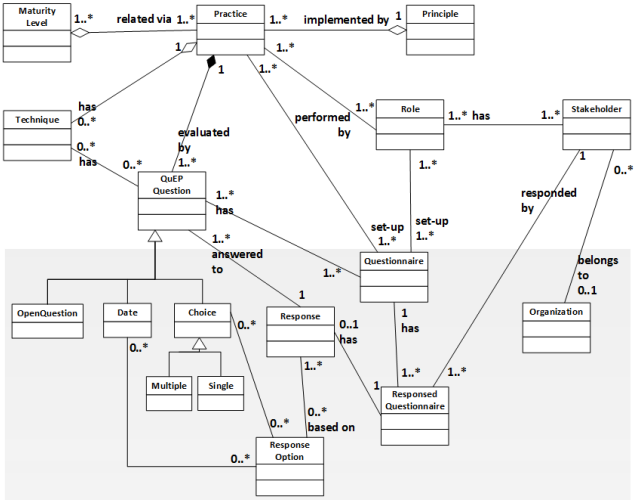


Figura 8. Modelo de diseño de QuEP-T

En este modelo de diseño se han incluido además nuevas entidades que representan explícitamente los cuestionarios y las respuestas de los participantes. Resaltado con fondo gris están las nuevas clases para gestionar los cuestionarios y la evaluación de la organización. En concreto, para cada una de las prácticas (*clase Practice*) se configuran cuestionarios (*clase Questionnaire*) formados por una serie de preguntas (*clase QuepQuestion*) asignadas a los participantes (*clase Stakeholder*) de la organización (*clase Organization*) según su rol (*clase Role*). Cada uno de los cuestionarios configurados tendrá un cuestionario de respuesta (*clase Responded Questionnaire*), que contendrá la respuesta realizada por parte de cada participante a una o más preguntas. El cuestionario contestado podrá contener una respuesta (*clase Response*) para cada una de las preguntas.

4.2 Arquitectura de QuEP-T

En la Figura 9 se presenta el diseño de la arquitectura, siguiendo una aproximación por capas (3 capas, 2 niveles). QuEP-T v1.0 ha sido desarrollada siguiendo el patrón de diseño MVC sobre una plataforma JEE 7, utilizando un servidor de aplicaciones Glassfish 4.1 en el entorno de desarrollo IDE Netbeans 8.1. Dentro del patrón MVC, la capa de presentación ha sido implementada utilizando el framework PrimeFaces 5.0, el cual está basado en tecnologías como JSF 2.2, AJAX, HTML5 y CSS3. Hemos seleccionado este framework, por la versatilidad de sus componentes que nos permiten presentar los diferentes informes de QuEP y sus gráficas de resultados de una manera más amigable para el usuario, brindando la flexibilidad de ajustarse a entornos web o a dispositivos móviles. En la capa de negocio y persistencia utilizamos EJB 3 and JPA Hibernate 4.3.1, respectivamente, facilitando la interacción con la información almacenada en la base de datos, gestionada por Postgres versión 9.5.1. Actualmente, QuEP-T v1.0 se encuentra desplegado en un servidor, proporcionando acceso a los usuarios del sistema (administrador y participantes), así como a los visitantes desde un navegador Web tal como Firefox o Chrome o desde cualquier dispositivo móvil.

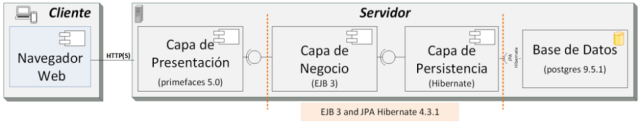


Figura 9. Arquitectura en capas para QuEP-T

4.3 QuEP-T v1.0

Finalizada la primera iteración, se han desplegado los cuestionarios para un usuario con un rol específico, configurados previamente a partir de la asignación de un conjunto de preguntas a cada una de las prácticas (ver Figura 10.a). Por otra parte, se ha desarrollado la presentación de resultados, mostrando inicialmente dos tipos de informes (ver Figura 10.b). En primer lugar, la presentación de los resultados obtenidos de forma global respecto de los niveles de madurez de QuEP; esta valoración se obtiene a partir de las

preguntas planteadas por cada principio/práctica. En segundo lugar, la presentación de los resultados obtenidos en cada nivel de madurez, disgregados por principios. En ambos casos se muestra el porcentaje alcanzado (completado/no completado). QuEP-T v1.0 permite que la organización pueda realizar su evaluación y conocer su nivel de madurez respecto al marco QuEP. Además de proporcionar un indicador de cómo mejorar a partir de los resultados (porcentaje no completado).

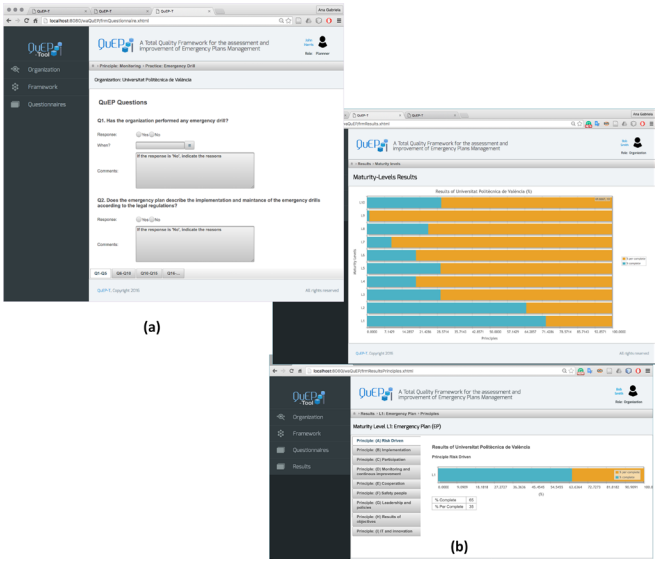


Figura 10. QuEP-T v1.0

5 Planificación del resto de iteraciones

Una vez realizada la primera iteración, y generada la versión 1.0 de QuEP-T, se han introducido 2 WUs (WU25, WU26) en el *product backlog* correspondientes a mejoras en la WU2. Tras la reordenación pertinente del *product backlog*, la tabla 4 muestra la planificación realizada para la iteración 2, y una planificación estimada del resto de iteraciones. Para cada iteración, se muestra las WUs incluidas y las tareas que el equipo ha definido. La duración de la iteración sigue siendo de 3 semanas. El momento de envío del presente trabajo se corresponde con el inicio de la segunda iteración del proyecto QuEP-T.

Tabla 4. Planificación de iteraciones

Iteración	ID	Nombre WUs/tarea	Estimación
Iteración 2 (3 semanas)	**WU25**	**Visualizar resultados obtenidos de la evaluación para una organización por prácticas**	**8**
	WU4	Configurar los tipos de preguntas.	4
	WU5	Modificación de tipos de preguntas	4
	WU6	Crear preguntas	4
	WU7	Modificar preguntas	4
	WU8	Configurar opciones de respuesta para cada pregunta	16
	WU9	Asignar roles, técnicas y características de resiliencia a las preguntas	8
	WU10	Creación de los cuestionarios dinámicos	24
	WU11	Modificación de los cuestionarios dinámicos	16
	WU12	Asignar usuarios a la organización	8
	WU13	Evaluar cuestionarios disponibles para un usuario según el rol	8
	WU14	Notificar evaluación de cuestionarios contestados	8
	Tarea 1	*Consultas e ingreso de nuevos datos en la Base de Datos*	*8*
Iteración 3 (3 semanas)	WU15	Visualizar las mejores prácticas	8
	WU26	**Visualizar resultados obtenidos de la evaluación para una organización por técnicas.**	**8**
	WU16	Notificar las técnicas que debe seguir la organización para mejorar su gestión.	8
	WU17	Creación de roles y usuarios.	8
	WU18	Modificación de usuarios y roles	8
	WU19	Borrado de usuarios y roles	4
	WU20	Crear componentes (niveles de madurez, principios, prácticas, técnicas y características resilientes)	24
	WU21	Modificar componentes (niveles de madurez, principios, prácticas, técnicas y características resilientes).	16
	WU22	Creación de organizaciones	8
	WU23	Modificación de organizaciones	8
	WU24	Borrado de organizaciones	4
	Tarea 1	*Mejora de las plantillas para las vistas*	*16*

6 Conclusiones y Trabajos Futuros

Aunque existen trabajos que se han centrado en la definición y mejora de modelos y métodos de planificación, no existe una herramienta de soporte basada en las TI, que permita a las organizaciones y planificadores medir la calidad de la gestión del plan de emergencia, más allá de una simple auditoría. En este sentido, QuEP proporciona un marco general para evaluar a cualquier organización respecto a su gestión de planes de emergencia, apoyado por una herramienta web, denominada QuEP-T. Su primera versión se ha desarrollado siguiendo un proceso de desarrollo iterativo basado en un enfoque ágil, en concreto SCRUM. Esto nos ha permitido incorporar cambios en la herramienta para atender a los nuevos requisitos, a la vez que tener una versión de la misma en un corto periodo de tiempo, facilitando así la implicación de los planificadores y, por tanto, el éxito de la propuesta. Las principales funcionalidades de QuEP-T v1.0 son la creación dinámica de los cuestionarios de evaluación, según el rol

del participante, y la generación de un informe preliminar de resultados que muestran su nivel de madurez según el marco QuEP, desglosado por principios.

Como trabajos futuros finalizaremos la herramienta QuEP-T siguiendo las iteraciones descritas en el presente artículo. Adicionalmente integraremos a nuestra herramienta características de resiliencia para contribuir a generar más valor al plan de emergencia y su gestión. Finalmente, pondremos QuEP-T a disposición de diversas organizaciones y evaluaremos su gestión de planes de emergencia.

Agradecimientos

El trabajo está parcialmente financiado por el proyecto CALPE (TIN2015-68608-R (MINECO/FEDER)), y el programa de becas SENESCYT-Ecuador.

Referencias

1. Norma Básica de Autoprotección (NBA), https://www.boe.es. (2007)
2. CPG101, Developing and Maintaining Emergency Operations Plans, FEMA. (2010)
3. Emergency response and recovery-UK. https://www.gov.uk
4. Guia para la elaboración de plan de emergencia y evacuacion (SIGWEB). http://www.sigweb.cl/biblioteca/GuiaPlanesEmergencias.pdf.
5. Emergency Management Planning Guide. Public Safety Canada. http://www.nce-rce.gc.ca
6. Emergency Response Management in Japan: Final Research Report, no. Asian Disaster Reduction Center, p. 47, (2011)
7. Núñez, A., Penadés, M.C., Canós, J.H., Borges, MR: Towards a Total Quality Framework for the Evaluation and Improvement of Emergency Plans Management. Proc. 13th Int. Conf. Inf. Syst. Cris. Response Manag. ISCRAM 2015. (2015)
8. Camisón, C., Cruz, S., González, T.: Quality management: concepts, approaches, models and systems. Pearson/Prentice Hall, Madrid; Santiago, Chile. (2007)
9. Dean, J.W., Bowen, D.E.: Management Theory and Total Quality: Improving Research and Practice through Theory Development. Acad Manag Rev. (1994)
10. EFQM.org. http://www.efqm.org/the-efqm-excellence-model.
11. JUSE.org. http://www.juse.or.jp/e/deming/.
12. Singhal/singhal. Implementing Iso 9001:2000 Qms: Reference Guide. (2008)
13. FUNDIBEQ, http://fundibeq.org/.
14. Berke, P., Godschalk, D.: Searching for the Good Plan: A Meta-Analysis of Plan Quality Studies. J Plan Lit. (2009).
15. Meyerson, R.: A Tool for evaluating plan quality of local Government emergency management response plans. http://www.mpa.unc.edu/sites/www.mpa.unc.edu/files/Rachel%20Meyerson.pdf. (2013)
16. Berke, P., Smith, G., Lyles, W.: Planning for Resiliency: Evaluation of State Hazard Mitigation Plans under the Disaster Mitigation Act. Nat Hazards Rev 13:139–149. (2012)
17. Nuñez, A-G., Penadés, M.C., Canós, J.H.: QuEP: Building a Continuous Improvement of Emergency Plans Management. Proc. 13th Int. Conf. Inf. Syst. Cris. Response Manag. ISCRAM 2016 (2016)
18. Charantimath, P.M.: Total Quality Management. Pearson Education India. (2011)
19. Camison, C.; Total quality management and cultural change: a model of organisational development. Int J Technol Manag 16:479–493. (1998)
20. Linstone H a., Turoff M (2011) Delphi: A brief look backward and forward. Technol Forecast Soc Change 78:1712–1719. doi: 10.1016/j.techfore.2010.09.011
21. Schwaber K, Sutherland J. The Scrum Guide. ScrumOrg and ScrumInc 17. http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf. (2013).

¿Qué desafíos presenta el desarrollo global del software? Aprende jugando

Aurora Vizcaíno[1], David Valencia[1], Juan Pablo Soto[2], Lilia García-Mundo[1] y Mario Piattini[1]

[1]Alarcos Research Group, University of Castilla-La Mancha, Ciudad Real, Spain
[2]Department of Mathematics, University of Sonora, Hermosillo, Mexico
{david.valencia1, liliacarmen.garcia}@alu.uclm.es,
{aurora.vizcaino, mario.piattini}@uclm.es, jpsoto@mat.uson.mx

Resumen—Las empresas de desarrollo de software intentan unirse al mercado global con el fin de poder contratar mano de obra en otros países, buscando reducir los costes, aumentar la productividad y así obtener ventajas competitivas. Esto es lo que se conoce como Desarrollo Global del Software (DGS o GSD, por sus siglas en inglés: Global Software Development). Para realizar esta práctica las empresas requieren desarrolladores que posean conocimientos y habilidades para solventar los problemas que surgen a causa de la distancia geográfica, temporal y cultural. Es aquí donde los juegos serios pueden jugar un papel importante, ya que se trata de juegos educativos que permiten adquirir conocimientos y habilidades con un bajo coste. En este artículo se describe un juego con el cual se puedan adquirir algunas de las competencias necesarias en el DGS. El juego simula escenarios que suelen presentarse durante el desarrollo global de un proyecto software, de manera que el usuario pueda tomar conciencia de los problemas referentes al DGS y adquirir una cierta experiencia a la hora de solventar estos problemas. Además, se describe una evaluación preliminar del mismo.

1 Introducción

En el área del desarrollo de software, la globalización ha llevado a muchas empresas a realizar el desarrollo de sus productos de una manera distribuida, llevándose a cabo por diferentes equipos, e incluso desde diferentes países. Este nuevo paradigma de desarrollo se conoce como "Desarrollo Global del Software" [1], el cual conlleva una gran cantidad de problemas adicionales al desarrollo de software tradicional.

Por ejemplo, la deslocalización de los equipos implica problemas de comunicación, coordinación y control, así como, aquellos derivados de las diferencias culturales de los distintos equipos [2]. Estos inconvenientes frecuentemente dificultan el entendimiento entre los participantes del proyecto, especialmente cuando éstos deben usar un lenguaje común (no nativo), pudiendo surgir malentendidos que afectan la comunicación y la coordinación del trabajo, y que podrían suponer un riesgo para el proyecto [3]. Otro aspecto importante es la falta de confianza que surge entre los miembros que participan en el DGS [4].

Por todo ello, es necesario que las personas que trabajan en el DGS posean competencias adicionales a las requeridas en el desarrollo tradicional.

Por lo general, es difícil encontrar un método adecuado para la enseñanza de estas habilidades, ya que las clases teóricas resultan insuficientes. Otros métodos, como el descrito en [5, 6], en el que estudiantes localizados en diferentes países llevan a cabo el desarrollo de un proyecto común, resultan costosos y complejos de coordinar.

Para preparar a los estudiantes o ingenieros en los desafíos que pueden encontrar en el desarrollo de un proyecto DGS se propone un juego serio que permita al usuario adquirir algunas de las competencias requeridas.

2 Un Juego Serio para el DSG

En este apartado nos centramos en describir la herramienta propuesta. En este caso el usuario jugará desempeñando el papel de un jefe de proyecto. El juego se basa en la planificación de un proyecto software, donde se simula trabajar con personas de distintas partes del mundo y el usuario tendrá que hacer frente a problemas que podrían presentarse en el DGS. Por ejemplo, la deslocalización de los equipos implica problemas de comunicación, coordinación y control, así como, aquellos derivados de las diferencias culturales de los distintos equipos [2]. Además de ser una herramienta que permite adquirir una serie de conocimientos, combina los aspectos esenciales de un juego, lo que proporciona un aprendizaje más entretenido y llevadero para el estudiante. El juego debía cumplir con una serie de requisitos que permitan simular escenarios que suelen presentarse cuando se trabaja en proyectos de DGS. Un escenario se compone de un nombre, una duración, un presupuesto para ese proyecto, los módulos que lo componen y los países que intervienen.

Algunas de las principales capacidades del juego se describen a continuación:

- El juego debe simular una serie de eventos o problemas inesperados que podrían presentarse cuando se participa en un proyecto GSD.
- El juego debe contar con distintos escenarios, los cuales tienen distintos niveles de dificultad. El usuario comenzará por el más sencillo e irá ascendiendo el nivel de dificultad.
- El juego simulará un chat, correo electrónico y teléfono para que el alumno utilice herramientas de comunicación síncrona y asíncrona, por lo que la aplicación permitirá simular aleatoriamente la llegada de emails, llamadas telefónicas y chats.
- El usuario debe poder elegir entre una lista de soluciones cada vez que en el escenario ocurra un evento inesperado, las cuales tienen una mayor o menor puntuación dependiendo de lo apropiadas que sean para resolver el problema en concreto.
- El juego dispondrá de un sistema de puntos, que fluctuarán según los días restantes para la entrega del software y el presupuesto disponible, de modo que una mala decisión por parte del usuario a la hora de enfrentar un evento

inesperado resultará en una pérdida del presupuesto y de días restantes que si la decisión seleccionada hubiese sido apropiada.

- El usuario podrá interactuar con empleados virtuales en los distintos escenarios. Los empleados se caracterizan por nombre, país, rol, salario, email, experiencia y una foto que los representa.

A. Herramienta

El juego que es una aplicación web, consta de dos subsistemas principales, uno para el estudiante y otro para el profesor que se encarga de proponer escenarios y supervisar al alumno.

Una vez que el estudiante decide jugar una partida, la aplicación le mostrará la interfaz principal del juego (Figura 1). Como se puede ver la interfaz se divide en tres columnas. La columna de la izquierda contiene información del proyecto (nombre, presupuesto, el tiempo que le queda para terminar el proyecto, la hora de los países involucrados, la confianza entre los miembros que trabajan en el proyecto, etc.).

Además es en esta columna en donde el estudiante puede acceder a la configuración de los módulos que componen el proyecto.

Fig 1. Interfaz principal del juego.

Una vez que los módulos estén configurados, podrá comenzar con la partida. La columna central contiene los botones para acceder al teléfono, chat y email, además, en esta columna, se muestra la información relativa a la acción que se esté ejecutando en cada momento. Por último, la columna de la derecha muestra la imagen del calendario, al que se puede acceder pulsando en dicha imagen (Figura 1), así como las distintas acciones que pueden llevarse a cabo durante el juego.

Durante la ejecución de la partida, al estudiante le irán apareciendo aleatoriamente problemas que suelen surgir cuando se trabaja en entornos de DGS, los cuales deberá ir solucionando conforme el juego avanza. Por ejemplo, un problema cultural, es decir, un trabajador de su equipo se queja de que su jefe es mujer. En este caso en particular, el sistema le solicita al usuario la recomendación que debería darle a su compañero ante tal situación. Esta problemática sigue presentándose en países donde el machismo es aún difícil de erradicar, por lo tanto, el usuario le debería de explicar y hacerle entender que en su país es una práctica habitual y por lo tanto, debería ser respetuoso ante dicha situación. Al final de cada partida, el sistema le mostrará al usuario el resultado obtenido durante la partida

Otra situación que se podría dar, por ejemplo, es la de recibir una llamada telefónica. El usuario deberá responder la llamada y atender el problema que se le plantee.

Por otra parte, existe el subsistema al que sólo puede acceder el profesor. En dicho subsistema el profesor puede crear problemas, llamadas de voz, chats, proyectos, ver resultado de la partida del estudiante, etc.

La herramienta ha sido evaluada por una experta en juegos serios siguiendo la métrica propuesta en [7]. La experta detectó oportunidades de mejora para algunos factores en los que se está trabajando actualmente.

3 Conclusiones y trabajo futuro.

En este artículo se describe un juego serio que sirve de apoyo para la adquisición de algunos de los conocimientos y habilidades que son necesarios en el DGS. Al ser un juego tiene la ventaja de ser mucho más asequible y entretenido que otros medios de formación tradicionales.

El juego se basa en la simulación de un escenario en el que se desarrolla un proyecto. El jugador debe conseguir desarrollar todas las fases que componen cada módulo.

Una vez concluido el proceso de evaluación se está trabajando en mejorar las funcionalidades siguiendo los comentarios de la experta que realizó la evaluación.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto GINSENG (TIN2015-70259) el proyecto LPS-BIGGER: Línea de productos Software para BiG Data a partir de aplicaciones innovadores en entornos reales (Ref.: UCTR150175.), se enmarca dentro del Programa estratégico CIEN, y es co-financiado por el Centro para el Desarrollo Tecnológico Industrial (CDTI), y Fondo Europeo de Desarrollo regional (FEDER).

Referencias

1. Herbsleb, J. D.; Moitra, D.: “Global software development”, IEEE Software, vol. 18(2), p. 16-2, 2001.
2. Vizcaíno, A; García, F.O.; Piattini, M. “Desarrollo Global de Software”, Ra-Ma, 2014.
3. Monasor,M.J.; Piattini, M.; Vizcaíno, A.: “Challenges and improvements in distributed software development: A systematic review”, Advances in Software Engineering, p. 14, 2009.
4. Moe, N.B and Smite,D.: "Understanding a lack of trust in Global Software Teams: a multiple-case study," *Softw. Process,* vol. 13, pp. 217-231, May, 2008.
5. Monasor, M.J.; Vizcaíno, A.; Piattini, M.: “Docencia en Desarrollo Global de Software: Una Revisión Sistemática”, en Jornadas de Enseñanza Universitaria de la Informática, p. 241-248, Sevilla, 2011.
6. Deiters, C.; Herrmann, C.; Hildebrandt, R.; Knauss, E.: Kuhrmann, M.:“GloSE-Lab: Teaching Global Software Engineering”, International Conference on Global Software Engineering, p. 156-160, 2011.
7. García-Mundo, L.; Genero, M.; Piattini, M.L “Refinamiento de un modelo de calidad para juegos serios”, Proceeding 2st Congreso de la Sociedad Española para las Ciencis del Videojuego, p. 68-79, Barcelona (Spain), June, 2015.

Smart Spaces: sistema de tecnoinclusión inteligente

Enrique Moguel[1], J. C. Preciado[1], Fernando Sánchez-Figueroa[1] y Juan Hernández[1]

[1] Quercus Software Engineering Group. Universidad de Extremadura
Av. de la Universidad, s/n. 10.071 (Cáceres)
[1]{enrique, jcpreciado, fernando, juanher}@unex.es

Resumen. En los últimos años las ciudades están evolucionando mejorando sus infraestructuras, y por consiguiente, facilitando y mejorando la vida de sus ciudadanos. Sin embargo, existe un sector de la sociedad más vulnerable, y que en muchas ocasiones están relegados a un segundo plano: las personas con diversidad funcional. Uno de los principales problemas es la accesibilidad, tanto arquitectónica, como urbanística, como en el transporte. Los problemas de accesibilidad no se limitan a la población minusválida, sino que se extiende a otros perjudicados como pueden ser personas con muletas o mujeres embarazadas. Smart Spaces es un sistema que facilita la integración de personas con movilidad reducida, facilitando el acceso a la información que nos rodea mediante una plataforma Web y aplicación para dispositivos móviles. Haciendo uso de una estrategia colaborativa para la adquisición de puntos de interés y mostrando dichos puntos a través de una capa de Realidad Aumentada, los ciudadanos con movilidad reducida podrán conocer si el punto de destino tiene plazas de aparcamiento adaptadas, podrán consultar su estado, podrán conocer la localización de las rampas de acceso, lavabos adaptados, ascensores, etc.

Palabras Clave: Smart Cities, Accesibilidad, Movilidad Reducida, Puntos de Interés, Realidad Aumentada.

1 Introducción y motivación

La movilidad, el acceso a edificios, la localización de distintos lugares de interés, etc., son acciones de la vida cotidiana que no suponen ninguna dificultad para personas que no tengan problemas de movilidad. Sin embargo, este tipo de acciones tan habituales se convierten en auténticas barreras para aquellos que tengan alguna discapacidad o movilidad reducida. En este marco, las TICs pueden aportar soluciones significativas para mejorar la calidad de vida del ciudadano y, especialmente, a aquellas personas que presenten algún tipo de diversidad funcional.

En esta línea, ya existen algunas soluciones que abordan esta problemática como *Accessibility*[1] *Wheelmap*[2] *Movilidad Donostia* o *Appcessibility*[3] Sin embargo, estas soluciones se enfocan a una región concreta (*Accessibility* –España–, *Wheelmap* –princilpamente Reuno Unido–), otras son poco escalables pues presentan dificultades notables para incluir nuevas regiones y nuevos puntos de interés *Movilidad Donostia* - San Sebastián-, o bien no cuentan con soporte y mantenimiento continuo (*Appcessibility)*. Adicionalmente, ninguna de estas aproximaciones presenta una solución colaborativa que permita actualizaciones continuas y en tiempo real de

[1] http://famma.org/accessibility
[2] http://wheelmap.org/
[3] https://itunes.apple.com/es/app/appcessibility/id499091763?mt=8

puntos de interés accesibles. De la misma forma, tampoco ofrecen soluciones de realidad aumentada que, haciendo uso de la cámara de nuestro dispositivo móvil, permitan una rápida localización de puntos de interés accesibles.

Para hacer frente a la problemática planteada en este tipo de soluciones, este artículo presenta *Smart Spaces*, un sistema de tecnoinclusión inteligente orientado al colectivo de personas con movilidad reducida, y que presenta de forma amigable y en tiempo real los distintos puntos de interés accesibles más cercanos a su destino: acceso a edificios, ascensores, lavabos adaptados, aparcamientos reservados, etc. *Smart Spaces* se basa en el uso intensivo de diferentes tecnologías web, soluciones de **cloud computing** [1] y **realidad aumentada** [2], para conseguir una aplicación multidispositivo orientada a facilitar el acceso a información relevante a personas con movilidad reducida.

Smart Spaces[4][5] ha sido galardonado con el premio **SISTEDES** y **UniDEVsity** en su edición de 2015 al mejor Trabajo Final de Máster sobre herramientas para el desarrollo de Apps, Games, Software y Aplicaciones Web[6].

El resto del artículo se estructura de la siguiente forma. En la sección 2 se propone la arquitectura en la que se basa *Smart Spaces* y en la sección 3 se presentan las conclusiones.

2 Smart Spaces: Arquitectura

La arquitectura de *Smart Spaces* fue diseñada con el objetivo de que cualquier usuario interesado en conocer puntos de interés accesibles en su entorno, pudiera acceder a la información y a la aplicación desde cualquier dispositivo con conexión a internet. Para conseguir este objetivo, la arquitectura (*Figura 1*) se estructuró en tres partes bien diferenciadas y que se detallan a continuación:

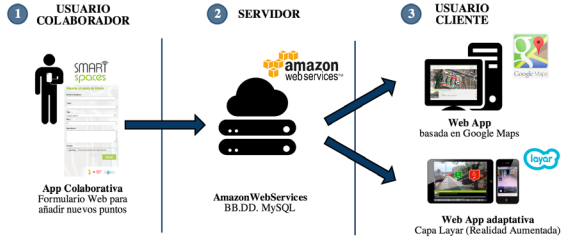


Figura 1.- Arquitectura del sistema Smart Spaces

1. **Usuario colaborador**: todos los puntos de interés, ya sean accesibles o no accesibles, se incluyen en el sistema de una manera colaborativa a través de un cuestionario Web. En este cuestionario se puede rellenar la información relevante relativa al punto de interés, a saber: nombre, tipo (acceso a edificio, ascensor, parking, escaleras, etc.), coordenadas (seleccionándose la posición en el mapa), piso (altura en interiores, en el caso de que se requiera), descripción y una imagen (por ejemplo una fotografía significativa).

[4] **Enlace proyecto:** http://uex.be/SmartSpaces
[5] **Enlace vídeo:** https://youtu.be/x-G-RgV_JkU
[6] **Enlace Web de SISTEDES:** http://goo.gl/sMRp53

2. **Servidor:** tanto los datos del sistema como la aplicación Web se alojan en la nube, haciendo uso de las tecnologías que ofrece AmazonWebServices. Los usuarios a través de la aplicación incluyen los puntos y su información relevante en una única Base de Datos MySQL. Gracias al almacenamiento de los datos y el alojamiento de la aplicación en la nube, cualquier usuario puede acceder a la misma desde cualquier dispositivo con conexión a internet.
3. **Usuario cliente**: el cliente ha sido desarrollado con WebRatio[7], una herramienta de modelado de aplicaciones Web multidispositivo que mejora la velocidad de desarrollo y la confiabilidad. El usuario puede acceder a la aplicación a través del navegador de su dispositivo (PC, Tablet o Smartphone) y acceder al sistema de visualización/navegación con vista aérea (*Figura 2*), aprovechando los sistemas que nos ofrece la API de Google Maps. Gracias a esta vista el usuario tiene **geolocalizados** todos los puntos de interés mencionados con anterioridad (tanto puntos accesibles como puntos no accesibles) marcándose, además, el estado de los mismos en tiempo real (por ejemplo, rampa en obras, ascensor fuera servicio, lavabo en primera planta, etc.). Cuando el usuario se encuentra ante un punto no accesible (por ejemplo escaleras), la aplicación le ofrece la ruta alternativa a su destino accediendo por puntos accesibles (por ejemplo rampas o ascensores). Adicionalmente, se ha desarrollado un sistema de realidad aumentada, que permite al usuario contextualizar dónde se encuentran esos puntos de interés en base a la imagen real que se puede percibir desde su dispositivo móvil (*Figura 3*). Esta solución diferencial puede ayudar a cualquier usuario, pero especialmente a los usuarios con distintos tipos de diversidad funcional para situar correctamente los mencionados puntos de interés.

Figura 2.- Vista Web de la aplicación Smart Spaces

Por último, nos gustaría señalar que este sistema nos servirá para recopilar todo tipo de datos e información, para realizar estudios y estadísticas que nos servirán para conocer mejor las necesidades de la sociedad y poder mejorar el sistema y toda su funcionalidad. Un ejemplo claro de esta funcionalidad sería estudiar el uso de las plazas de aparcamiento adaptadas y valorar si las plazas están siendo sobreutilizadas (necesidad de adaptar nuevas plazas) o infrautilizadas (no serían necesarias tantas plazas adaptadas).

[7] http://www.webratio.com/site/content/es/web-application-development

Figura 3.- Vista de la capa de Realidad Aumentada en dispositivos móviles

3 Conclusiones

La solución desarrollada y propuesta en este artículo consiste en ofrecer un sistema **tecnoinclusivo** de información en tiempo real de la ubicación y estado de los distintos puntos de interés para las personas con algún tipo de discapacidad relacionada con la movilidad.

La aplicación se ha creado y testeado para el entorno de la Universidad de Extremadura en el campus de Cáceres. El sistema puede ser fácilmente implantado en otros centros de los diferentes campus de esta Universidad, facilitando así los desplazamientos *inter*/*intra* campus de los estudiantes (por ejemplo, cuando necesite acceder a la biblioteca o asistir a un curso de formación en un centro de otro campus). Del mismo modo, la escalabilidad del proyecto asegura sus posibilidades de implantación, no sólo en otras universidades, sino también en entornos diferentes al universitario (por ejemplo Administraciones Públicas, Centros Comerciales, zonas turísticas, etc.).

Gracias a toda esta información que proporciona la comunidad universitaria, la Universidad de Extremadura conocerá "a pie de campo" las deficiencias en sus infraestructuras para ofrecer una solución.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad (TIN2015-6957-R), la Junta de Extremadura (GR15098) y los Fondos FEDER.

Referencias

[1] Lizhe Wang; Jie Tao; Marcel Kunze; Alvaro Canales Castellanos; David Kramer; Wolfgang Karl. *Scientific Cloud Computing: Early Definition and Experience*. High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International. Pp. 825-830, 25-27 Sept. 2008.

[2] Enrique Moguel; M. Á. Preciado; J. C. Preciado: *A Smart Parking Campus: An Example of Integrating Different Parking Sensing Solutions into a Single Scalable System*. ERCIM News 98. July 2014, Special theme: Smart Cities, pp. 29-30.

SCIE
SOCIEDAD CIENTÍFICA INFORMÁTICA DE ESPAÑA

VNiVERSiDAD D SALAMANCA
CAMPUS DE EXCELENCIA INTERNACIONAL

GRUPO DE INVESTIGACIÓN BISITE

V Congreso Español de Informática
Salamanca, 13 al 16 de septiembre, 2016

CEDI 2016

ISBN: 978-84-9012-627-1 (PDF)