V SIMPOSIO DE SISTEMAS DE TIEMPO REAL

J. Javier Gutiérrez y Michael González Harbour (eds.)

STR2016

AQUILAFUENTE

Salamanca
University Press

V SIMPOSIO DE SISTEMAS DE TIEMPO REAL

J. JAVIER GUTIÉRREZ Y MICHAEL GONZÁLEZ HARBOUR (Eds.)

V SIMPOSIO DE SISTEMAS DE TIEMPO REAL

Ediciones Universidad
Salamanca

AQUILAFUENTE
223

Motivo de cubierta:
Diseñadora María Alonso Miguel

1.º edición: septiembre, 2016
ISBN: 978-84-9012-631-8 (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 Universitarias Españolas
www.une.es

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

Mensaje del Comité Organizador

Con este mensaje os damos la bienvenida a la V edición del Simposio de Sistemas de Tiempo Real que se celebra en el marco del Congreso Español de Informática, CEDI 2016. Este evento, que se lleva organizando desde la primera edición del CEDI, es una oportunidad para que la comunidad española de sistemas de tiempo real pueda dar visibilidad a sus últimos trabajos en un entorno más amplio y heterogéneo que el de las Jornadas de Tiempo Real que se vienen organizando anualmente de manera ininterrumpida desde 1998, y en este año 2016 han cumplido su XIX edición.

En esta ocasión, tenemos la fortuna de abrir el programa con la intervención de Francisco J. Cazorla, investigador del CSIC, que impartirá una charla invitada sobre uno de los temas que más interés y oportunidades de investigación está despertando en el área de tiempo real: los procesadores multicore. Francisco J. Cazorla es investigador del CSIC en el BSC (*Barcelona Supercomputing Center*), donde dirige el grupo CAOS (*Computer Architecture Operating System Interface*). Su trabajo se centra principalmente en sistemas computadores con énfasis en el diseño de hardware para tiempo real y alto rendimiento, y en técnicas de análisis temporal.

El programa técnico cuenta con nueve trabajos de gran calidad que han sido revisados por el Comité de Programa, y que aparecen en estas actas agrupados por los siguientes temas: (1) middleware de comunicaciones y casos de estudio, (2) sistemas operativos y gestión de recursos, y (3) modelado, análisis temporal, configuración y optimización.

Desde la organización nos gustaría expresar nuestro agradecimiento al conferenciante invitado y a todas las personas que con sus presentaciones han contribuido al programa del simposio. También queremos agradecer a todos los participantes su presencia y aportaciones, así como a la organización local por facilitar la celebración del evento. Esperamos que el programa sea de vuestro agrado.

Un afectuoso saludo del Comité Organizador del V Simposio de Sistemas de Tiempo Real

J. Javier Gutiérrez y Michael González Harbour

Organización

V Simposio de Sistemas de Tiempo Real

Comité organizador

J. Javier Gutiérrez García	Universidad de Cantabria
Michael González Harbour	Universidad de Cantabria

Comité de programa

Alejandro Alonso Muñoz	Universidad Politécnica de Madrid
Bárbara Álvarez Torres	Universidad Politécnica de Cartagena
Patricia Balbastre Betoret	Universidad Politécnica de Valencia
Manuel Capel Tuñón	Universidad de Granada
Alfons Crespo Lorente	Universidad Politécnica de Valencia
Juan Antonio de la Puente Alfaro	Universidad Politécnica de Madrid
Manuel Díaz Rodríguez	Universidad de Málaga
José María Drake Moyano	Universidad de Cantabria
Elisabet Estévez Estévez	Universidad de Jaén
Marisol García Valls	Universidad Carlos III de Madrid
Michael González Harbour	Universidad de Cantabria
J. Javier Gutiérrez García	Universidad de Cantabria
Marga Marcos Muñoz	Universidad del País Vasco
Pau Martí Colom	Universidad Politécnica de Catalunya
Joan Vila Carbó	Universidad Politécnica de Valencia
José Luis Villarroel	Universidad de Zaragoza
Juan Zamorano Flores	Universidad Politécnica de Madrid

Entidades colaboradoras:

ÍNDICE

Conferencia invitada

Conferencia invitada

Title: *Time Predictability and Composability in High-Performance Mixed-Criticality Multicore Systems*

Francisco J. Cazorla
BSC (Barcelona Supercomputing Center)

Abstract: The fundamental paradigms for the definition of Critical-Real Time Embedded Systems (CRTES) architectures are changing due to cost pressure, flexibility, extensibility and the demand for increased functional complexity. CRTES have been based on the federated architecture paradigm, which simplifies verification by providing a separation of responsibilities, hence enabling each provider to implement the hardware and software for a particular function independently from other suppliers. However, implementing an increasing amount of functionality on a Federated Architecture requires a high number of hardware units, making federated implementations inefficient in terms of size, weight and power consumption.

To cope with such problem, the automotive and avionics industries are adopting Integrated Architectures (IA). One fundamental requirement of integrated architectures is to ensure that incremental qualification (verification) is possible, whereby each software partition can be subject to verification and validation – including timing analysis – in isolation, independent of the other partitions, with obvious benefits for cost, time and effort.

In this talk I will focus on the timing component of incremental qualification. I will present some hardware support that can enable composability while providing high performance. I will also talk about existing software support to increase time predictability and time composability. I will also talk about the feasibility of a probabilistic timing analysis approach, and the hardware support required in order to enable it, as a way to provide high performance and time composability.

Biography: Francisco J. Cazorla is a researcher at the National Spanish Research Council (CSIC). He is currently the leader of the group on Interaction between the Operating System and the Computer Architecture at Barcelona Su- percomputing Centre (http://www.bsc.es/caos). His research area focuses on multithreaded architectures for both high-performance and real-time systems. He has co-authored over 100 papers in international refereed conferences. He has participated in several projects with industry including several processor vendor companies (IBM, Sun microsystems) and European FP6 (SARC) and FP7 Projects (MERASA, parMERASA, SAFURE, PROARTIS). He currently leads the PROXIMA project and one project with the European Space Agency.

Middleware de comunicaciones y casos de estudio

Un Middleware Centrado en Datos para el Control en Tiempo Real de Redes de Energía Inteligentes

Jaime Chen[1], Eduardo Cañete[1], Manuel Díaz[1], Daniel Garrido[1], y Krzysztof Piotrowski[2]

[1]Universidad de Málaga, Málaga, España
{hfc, ecc, mdr, dgarrido}lcc.uma.es
[2]IHP, Frankfurt (Oder), Alemania
piotrowski@ihp-microelectronics.com

Resumen. Este artículo presenta un middleware centrado en datos que permite en tiempo real la comunicación y almacenamiento de datos en redes eléctricas inteligentes. El middleware ha sido diseñado teniendo en cuenta la heterogeneidad de dispositivos y plataformas software utilizadas. Para su utilización, se dispone de una interfaz de datos que permite la comunicación entre las diferentes partes del sistema a través de servicios web REST (Representational State Transfer), así como una interfaz funcional que permite realizar operaciones de más alto nivel. El middleware está siendo utilizado para la implemenación de los demostradores del proyecto europeo e-balance, y ha tenido en cuenta para su utilización, la presencia de requisitos blandos de tiempo real, aspectos de seguridad y la escasa capacidad de algunos de los dispositivos en los que tendrá que ejecutarse.

Palabras clave: Middleware, Sistema Empotrado, Smart Grid, Control

1 Introducción

Las redes eléctricas inteligentes, también conocidas como *Smart Grids*, tienen una importancia cada vez mayor en el concepto más amplio conocido como ciudades inteligentes o *Smart Cities*. Cada vez es más habitual disponer por parte de las compañías eléctricas o intermediarios, de mecanismos para conocer en tiempo real el consumo por parte de sus clientes. A su vez, los clientes tienen nuevas posibilidades en forma de aplicaciones para móviles o tabletas donde pueden conocer, también en tiempo real, el estado de su facturación.

Por otra parte, asistimos también al auge de los llamados electrodomésticos inteligentes: lavadoras, lavaplatos o incluso frigoríficos. Todos estos dispositivos pueden estar conectados, bien a través de redes propias, bien a través de Internet, de manera que es posible monitorizarlos y controlarlos de manera remota.

Todas estas nuevas posibilidades, llevan de manera implícita, el manejo de una gran cantidad de datos a distintos niveles: dispositivos, aplicaciones de usuario, estaciones eléctricas, suministradores, etc. Hay que tener en cuenta, además, no sólo la

comunicación de los datos, sino también su almacenamiento, tanto para poder ser transmitidos a su vez a otros dispositivos o elementos de la red eléctrica inteligente, como para la gestión de la red mediante algoritmos inteligentes. Todo ello en tiempo real y con dispositivos de capacidades muy diferentes: desde sensores, pasando por pequeños mini-PC (p.ej. BeagleBones) o servidores con mayores capacidades de cómputo. Existe, asimismo, un amplio conjunto de aplicaciones para los diferentes roles que podemos encontrar: clientes, suministradores, operadores de la red, etc.

Este artículo se enmarca en el proyecto europeo del 7º programa marco, e-balance [1]. El principal objetivo del proyecto es diseñar e implementar una infraestructura basada en algoritmos eficientes capaz de gestionar la red de una manera inteligente.

Desde el punto de vista del usuario final, e-balance será capaz de analizar y controlar la producción y consumo de energía de los productores/consumidores, tomando decisiones inteligentes que afecten al comportamiento de los usuarios y logren un uso mejor de la energía. Desde el punto de vista de la infraestructura, e-balance pretende desarrollar un sistema capaz de intercambiar información de manera eficiente (tiempo real blando) entre todos los dispositivos que forman parte del mismo. Para lograr esta comunicación, se ha desarrollado un middleware encargado tanto de las comunicaciones como del almacenamiento distribuido de los datos.

El proyecto, con una duración de 42 meses, se encuentra ya en su tercer año, en los que se realizará el despliegue del sistema en dos demostradores con usuarios reales, uno de ellos situado en la localidad de Batalha (Portugal) y otro en un parque de ocio situado en Bronsbergen (Países Bajos).

Este artículo se centra en la presentación del middleware desarrollado para lograr llevar a cabo los objetivos de control y de gestión de la energía del proyecto.El sistema desarrollado es altamente distribuido y heterogéneo, con una gran cantidad de dispositivos, muchos de ellos empotrados, ejecutándose en paralelo y teniendo que cooperar entre ellos para lograr llevar a cabo una ejecución eficiente y de tiempo real del sistema. Los requisitos de tiempo real del mismo, no son altamente exigentes en cuanto a periodos o plazos (tiempos en el orden de segundos), pero sí es muy importante mantenerlos, dado que está en juego un funcionamiento adecuado de la red eléctrica, vital en la vida de hoy día.

La estructura del artículo es como sigue: la sección 2 presenta la arquitectura del proyecto e-balance. En la sección 3 se detalla el diseño del middleware y los componentes que lo forman. La sección 4 presenta la interfaz de datos que puede utilizarse para conectar a las diferentes unidades. La sección 5 presenta algunos detalles de implementación. Finalmente, se presentan algunas conclusiones y trabajos futuros.

2 Arquitectura de e-balance

El proyecto e-balance presenta algunas características típicas de los sistemas fractales. En esta visión fractal, las soluciones que se aplican en un nivel (p.ej. una casa), pueden luego volver a aplicarse en un nivel superior (p.ej. un vecindario) y así sucesivamente (p.ej. una ciudad). Esta visión fractal simplifica en cierta manera el desarrollo de e-balance y tiene una gran influencia sobre su arquitectura.

En el estado actual del proyecto, la arquitectura de e-balance se compone de 3 niveles: HAN, LV-FAN y MV-FAN correspondientes, respectivamente, a los niveles del cliente, estación de bajo voltaje (o secundaria) y de medio voltaje (o primaria).

En la Fig. 1 los componentes de e-balance están representados por las formas en color azul oscuro. Las cajas en azul claro representan el nivel de generación y transmisión de la red.

El sistema formado por e-balance incluye las denominadas unidades de gestión (*Management Units* o MUs). La figura muestra la relación jerárquica entre los diferentes niveles de e-balance con una MU para cada nivel.

En esta arquitectura, el middleware se ejecuta en las MU, recibe los datos desde las capas inferiores de comunicaciones, se encarga de gestionar los aspectos de privacidad y seguridad, y procede a almacenar los datos localmente hasta que estos sean solicitados por otras partes del sistema. Es necesario tener en cuenta también la interacción con sensores y actuadores.

Las formas en amarillo representan la red eléctrica y los dispositivos que hay en ella. Finalmente, las cajas en rojo representan marcos virtuales como el mercado, acceso a datos globales y operaciones. Las diferentes líneas en la figura representan varias clases de interacciones posibles entre los módulos que conectan.

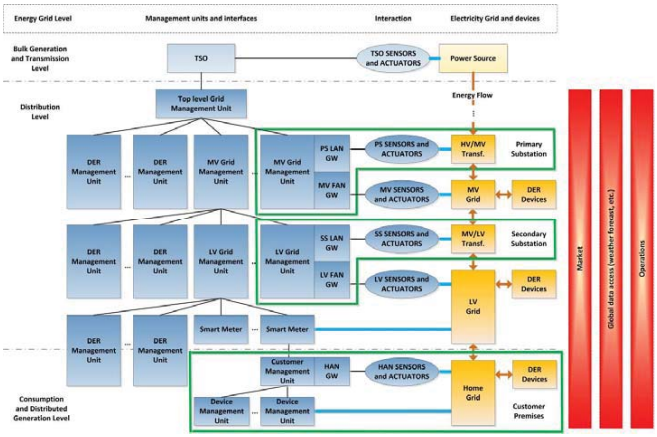


Fig. 1. Arquitectura de e-balance

Todas las MU tienen una arquitectura similar [2]. Sin embargo, dependiendo del nivel, pueden tener diferentes roles y obligaciones. Los procesos ejecutados en ellas pueden operar y procesar datos de diferentes partes, pero dada la arquitectura fractal de e-balance, los algoritmos de gestión aplicados en los diferentes niveles comparten la misma base conceptual, lo que mejora la escalabilidad de la propuesta.

El nivel de los dispositivos es el más bajo representado en la arquitectura. Un dispositivo puede ser, típicamente, un electrodoméstico que sólo consume energía, pero también puede ser una unidad de generación o almacenamiento. La denominada *Device Management Unit* (DMU) es la unidad central del dispositivo, que está al tanto de su estado y permite, además, su control. Para poder controlar las DMU, disponemos de las denominadas *Customer Management Unit* (CMU), que han sido diseñadas teniendo en cuenta la enorme heterogeneidad de dispositivos. Para ello, las CMUs utilizan plugins que pueden incorporarse dinámicamente. Por ejemplo, actualmente el consorcio está en contactos con una conocida empresa de electrodomésticos. La comunicación con estos dispositivos puede realizarse a través de Internet con una API proporcionada por esta empresa. Las CMUs están equipadas también para comunicarse con los sensores y actuadores de la denominada *Home Area Network* (HAN), red local de una casa, que interactúan con la red inteligente proporcionando monitorización, control, y, sobre todo, automatización.

El nivel por encima de las CMUs es el de la red eléctrica de bajo voltaje, donde encontramos las LVGMU (*Low Voltage Grid Management Unit*). Estas MUs están localizadas en las subestaciones secundarias, y cada una de ellas, controla los sensores, actuadores, CMUs y recursos de energía distribuidos (DER) de su nivel. Aquí, se repite la arquitectura, una LVGMU está equipada para comunicarse, en la denominada LV-FAN (*Low Voltage – Field Area Network*) con los niveles inferiores y también con el nivel superior, así como con los sensores y actuadores situados en el transformador de tensión media a baja y con los alimentadores de la estación secundaria.

Una MVGMU (*MediumVoltage Grid Management Unit*) es similar a su equivalente de bajo voltaje. Están situadas en las subestaciones primarias y equipadas para comunicarse con sensores, actuadores y LVGMU por debajo de la MU, en la denominada MV-FAN (*Medium Voltage – Field Area Network*).

La denominada TLGMU (*Top Level Grid Management Unit*) controla todas las MVGMU y DER conectados directamente a la red de medio voltaje. La TLGMU puede ser considerada como un centro de control que proporciona interfaces para las herramientas de gestión (sistemas SCADA), gestión de mercado, cortes, y otras.

Los párrafos previos dan una idea de la complejidad del sistema e-balance desde el punto de vista de las comunicaciones: está compuesto por diferentes dispositivos hardware (MUs, sensores, medidores inteligentes) con diferentes recursos, sistemas operativos y hechos por diferentes fabricantes. Además, todos estos dispositivos, tienen que comunicarse de manera segura entre sí, y con requisitos de tiempo real, generosos en cuanto a plazos (en el orden de segundos), pero existentes, al fin y al cabo.

3 Diseño del Middleware

El middleware presentado en este artículo se ha desarrollado con el objetivo de facilitar el desarrollo de aplicaciones en e-balance proporcionando un marco de abstracción sobre los detalles de las comunicaciones. Como se ha comentado, en nuestro sistema el

middleware se ejecutará en las diferentes unidades de gestión: TLGMU, MVGMU, LVGMU, DERMU y CMU. Los tres objetivos principales del middleware son:

1. Proporcionará medios para el intercambio de información: el middleware manejará automáticamente los niveles de red y proporcionará una manera simple de comunicación entre los dispositivos.
2. Proporcionará medios para almacenar/recuperar información: a través de una API. Los dispositivos pueden obtener y almacenar información persistente desde/de dispositivos remotos o desde/de el dispositivo local.
3. Elevar el nivel de abstracción sobre el que estos dispositivos están programados: el middleware proporciona una API simple de acceso a datos basada en un esquema de comunicación *publish/subscribe*.

3.1 Arquitectura del Middleware

La **Fig. 2** muestra la arquitectura de la plataforma de comunicaciones y su relación con la infraestructura de energía subyacente. La plataforma de comunicaciones se encuentra sobre las pilas de protocolos de red y por debajo de las aplicaciones e-balance. La plataforma de comunicaciones hace uso de un nivel de adaptación llamado *Communication Manager Module* que adapta las diferentes soluciones de red a un mismo "lenguaje" común para el middleware. El módulo de persistencia de datos (*Data Persistence Module*) almacena información y proporciona medios para almacenarla y recuperarla. El módulo de gestión de grupos y mantenimiento (*Maintenance and Group Management Module*) gestiona la red y la relación entre dispositivos. El módulo de procesamiento de solicitudes (*Request Processor Module*) se encarga de manejar todas las peticiones recibidas por la plataforma de comunicaciones. Este módulo es también responsable del procesamiento adecuado de las solicitudes de acuerdo con las políticas de seguridad y privacidad de los propietarios de los datos. Esta funcionalidad está implementada en el submódulo de control de acceso a datos (*Data Access Control Submodule*).

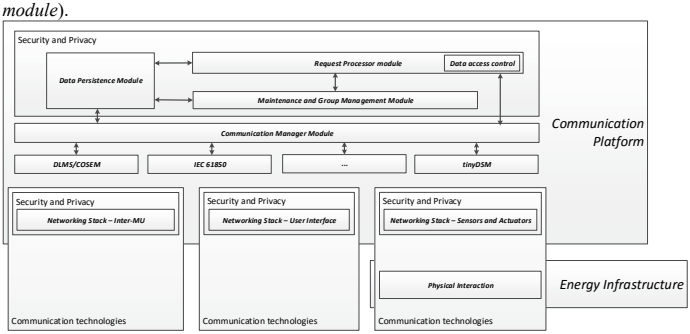


Fig. 2. – Arquitectura del middleware de comunicaciones de e-balance

3.2 Diseño software

La **Fig. 3** muestra una visión general del middleware desde el punto de vista del software. El diagrama muestra los diferentes módulos que conforman el middleware (persistencia de datos, procesamiento de peticiones, gestión de grupos y mantenimiento, control de acceso a datos y gestión de usuarios) y los servicios proporcionados por cada uno de ellos. Existe también una librería adicional denominada *EBCommon*, que se usa de manera interna para facilitar el desarrollo del middleware con funciones de apoyo y otra funcionalidad tales como manejo de tiempo, interfaz funcional para otros módulos, etc.

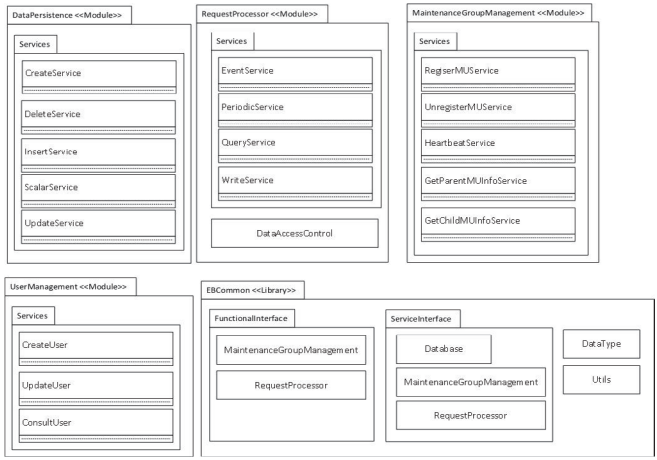


Fig. 3. – Diagrama de paquetes

El middleware está formado por un conjunto de módulos independientes que colaboran juntos a través de interfaces (servicios proporcionados por cada módulo). Esta alternativa de diseño permite a los diferentes módulos ser reemplazados sin afectar a los otros módulos. Por ejemplo, si se tiene que utilizar una nueva clase de base de datos, entonces, sólo se tiene que reemplazar el módulo de persistencia de datos. Es tarea del desarrollador adaptar la nueva base de datos a la API proporcionada por el módulo de persistencia de datos para hacer que sea compatible con el resto de módulos.

4 Interfaz de Datos de las Unidades de Gestión

La interfaz de datos (*Data Interface*) define las funciones a través de las cuales otros módulos software situados en la MU local o en otra remota pueden interactuar con una MU específica. Un ejemplo de este tipo de módulos es la Plataforma de Gestión de Energía de e-balance, que es la responsable de la ejecución de los algoritmos de balanceo de energía. La interfaz de datos permite interactuar con una MU específica leyendo y/o escribiendo datos desde/a ella. El principal objetivo de esta interfaz es hacer más fácil el desarrollo de aplicaciones en e-balance.

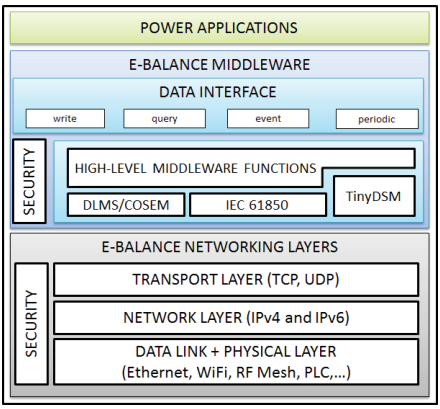


Fig. 4. – Contexto de la interfaz de datos (Data Interface)

El principal objetivo de diseño de la interfaz de datos (**Fig. 4**) fue la simplicidad Por esta razón, se compone únicamente de cuatro operaciones básicas:

- **Query:** esta operación permite leer valores de una variable específica de e-balance en un momento dado.
- **Write:** operación para actualizar/modificar el valor de una variable.
- **Event:** esta operación permite monitorizar una variable e-balance y recibir notificaciones en caso de que se cumplan determinadas condiciones sobre el valor de la misma (p.ej. superar un límite umbral).
- **Periodic:** esta operación permite subscribirse para recibir periódicamente los valores de una variable de e-balance. Un ejemplo puede ser la lectura periódica de una variable de consumo para refrescar los valores de la misma en la GUI de un cliente. La diferencia con respecto a la subscripción a eventos es que aquí no hay condiciones basadas en el valor de la variable, en su lugar, se define un periodo entre notificaciones.

Una operación puede únicamente ser realizada sobre variables existentes en el sistema si quien realiza la invocación tiene los permisos requeridos. La lista de variables existentes está predefinida dentro de e-balance.

Toda la comunicación realizada en el sistema se realiza únicamente a través de esta API. El middleware automáticamente convierte las llamadas a la API a los mensajes correspondientes en la plataforma de red específica que se esté usando. Los eventos y las operaciones periódicas siguen el paradigma de comunicación *publish/subscribe*. La aplicación se subscribe a una variable y los eventos son enviados cuando se actualice el valor o se cumpla alguna condición. Además de esta forma de comunicación, la aplicación también puede realizar peticiones bajo demanda con las operaciones de lectura y escritura.

5 Detalles de Implementación

En esta sección se describen los aspectos de implementación del middleware tal y como han sido aplicados en el proyecto e-balance.

5.1 Invocaciones remotas con ServiceStack

El middleware ha sido implementado utilizando ServiceStack [3], que es un marco de trabajo de código abierto diseñado para crear servicios web bajo el entorno .NET. ServiceStack permite modularizar la funcionalidad del middleware y estructurarlo en 2 niveles: *plugins* y servicios.

La funcionalidad básica del middleware se ha realizado con los servicios, y los *plugins* actúan como módulos capaces de contener tantos servicios como se requiera.

El concepto de *plugin* ha sido utilizado para implementar todos los módulos descritos en la sección 3.2.

Cada servicio web ServiceStack es modelado a través de un *Request DTO* (*Data Transfer Object*) y un *Response DTO*. Esto es, la entrada al servicio es un *Request DTO* y la salida es un *Response DTO*. Ambos *DTOs* definen las interfaces del servicio. Viendo el servicio como una función, el *Request DTO* podría representar a los argumentos de entrada y el *Response DTO* podría ser el conjunto de argumentos de salida.

5.2 Variables e-balance

Todos los datos de e-balance son modelados como tablas de una base de datos. Una variable e-balance es una tabla de datos donde el nombre de la tabla se refiere a la variable, y los campos de la tabla hacen referencia a las propiedades de la variable. Una variable e-balance puede tener tantas propiedades como haga falta, pero por defecto, hay tres propiedades obligatorias: *Id*, *Timestamp* y *Value*.

También podría ser posible tener todas las variables de e-balance en una tabla, pero la efectividad de la aproximación depende de la arquitectura subyacente del módulo de base de datos. En este caso, tener todas las variables en una tabla permitiría añadir nue-

vas variables más fácilmente, pero esto también puede hacerse en la aproximación elegida, si fuera necesario (las variables son predefinidas). Por otra parte, la principal ventaja de la aproximación utilizada es que tener una tabla por variable permite separar los datos con diferentes significados/funciones.

Supongamos que necesitamos una variable para almacenar el consumo de energía en el sistema. Para lograr, esto existe una tabla como la mostrada en la Tabla 1 que puede ser creada y usada para tal fin.

ENERGY_CONSUMPTION		
Id	**Timestamp**	**Value**
1	12/02/2015 14:10:00	30.4
2	12/02/2015 14:20:00	25.8

Tabla 1. Ejemplo: Tabla de consumo de energía

La tabla representa la variable e-balance llamada *energy_consumption*. Esta variable tiene las siguientes propiedades.

- **Id**: identificador de una tupla específica – la estructura de datos representando un valor simple de la variable.
- **Timestamp**: instante de tiempo del valor *energy_consumption* almacenado, el identificador temporal del valor.
- **Value**: el valor de la variable *energy_consumption* para cada punto en el tiempo.

Para trabajar con esta variable, la implementación proporciona una estructura de datos denominada `EBVariable` con los siguientes campos.

- **Name**: se refiere al nombre de la variable, p.ej. `energy_consumption`.
- **Properties**: propiedades de una variable e-balance específica, p.ej. `[Time, Value]`
- **Values**: contiene los valores que van a ser almacenados/leídos desde una variable e-balance.
- **Condition** – permite definir condiciones tipo SQL para leer conjuntos de tuplas que satisfagan una condición, p.ej., `"Id >2 and Id<5"` o `"Timestamp = 12/02/2015 14:20:00"`. Se basa en las propiedades definidas.

5.3 Interfaz de servicios web RESTful

Como se describió anteriormente, la interfaz de datos del middleware se compone de cuatro operaciones básicas. Cada una de estas operaciones (`write, query, event` y `periodic`) está implementada utilizando un servicio web RESTful a través del cual una aplicación puede interactuar fácilmente con una MU a través de Internet.

Representational State Transfer (REST) define un conjunto de principios arquitecturales a través de los cuales se pueden diseñar servicios web que sigan estos principios. Típicamente, un servicio web que utilice REST (un servicio RESTful) utiliza HTTP como su protocolo subyacente.

La utilización de los servicios web hace posible, en general, comunicarse con las MU desde prácticamente cualquier clase de hardware, sistema operativo y lenguaje de programación.

Cada servicio web tiene una forma de solicitud definida, así como una respuesta. Con ServiceStack esto se hace a través de los *DTO*. Por ejemplo, para la operación de escritura usaríamos el siguiente *DTO* para realizar solicitudes (ejemplo en Fig.5):

- **RequestSource:** identificador de quien realiza la llamada; dispositivos y servicio/participante. La URL del servicio web está compuesta de los siguientes campos:
  - **Protocol**: puede ser *http* o *https*.
  - **IP**: Dirección IP de la MU donde el servicio web está localizado.
  - **Port**: el Puerto a través del cual el servicio web puede ser accedido.
  - **ServicePath**: ruta dentro de la MU donde el servicio web está localizado.
- **Variable**: variable que va a ser almacenada.
  - **Name:** nombre de la variable e-balance.
  - **Properties**: indica las propiedades exactas de la variable e-balance que va a ser almacenada o modificada. Si su valor es nulo, el parámetro `Values`, debe contener un valor para cada propiedad.
  - **Condition**: este parámetro debe establecerse a nulo cuando vaya a insertarse un nuevo valor. Sin embargo, será útil para modificar tuplas existentes, ya que, permitirá identificarlas.
- **Values:** define los valores de la variable que va a ser almacenada o modificada.

La respuesta sería también a través de otro *DTO*:

- **OperationResults**:
  - **OpCode**: indica un código para describir el resultado de la operación de una forma concisa.
  - **Info**: proporciona un mensaje descriptivo sobre la operación.
  - **Success**: indica si el servicio se ejecutó de manera satisfactoria.

Data Owner			
Protocol	**IP**	**Port**	**ServicePath**
http	192.168.43.98	2554	/

e-balance Variable		
Name	**Fields**	**Condition**
ENERGY_CONSUMPTION	Time, Value	Null

Values
12/04/2015 12:43:00, 23.2

Fig. 5. Ejemplo de DTO para operación de escritura

5.4 Interfaz de Datos Funcional

Todas las funciones proporcionadas por la interfaz de datos están basadas en servicios web RESTful, de manera que dos MUs pueden interactuar entre ellas o intercambiar datos a través de servicios web ejecutados sobre IP. Esta clase de interfaz es bastante restrictiva en el sentido de que sólo permite hacer a los usuarios direccionamiento *unicast*. Se se ha desarrollado también una interfaz funcional que permite, no solamente hacer *broadcast* (p.ej. una LVGMU podría solicitar información de varias CMUs), sino que mejora también la utilidad de la interfaz de datos al poder ser utilizada desde otros módulos internos de e-balance (aplicaciones e-balance). La **Tabla 2** muestra ejemplos de operaciones que pueden utilizarse con esta API funcional.

Operación	Cabecera método
Escritura de valores unicast en una MU específica	`WriteResponse Write(EBUrl destinationMU, EBVariable ebVariable)`
Escritura de valores en varias MUs especificadas por el parámetro destinationMU	`WriteResponse[] Write(EBUrl[] destinationMUs, EBVariable ebVariable)`
Escritura de varias variables en todas las MUs hijas de la que hace la llamada	`WriteResponse[] WriteInChildMUs(EBVariable ebVariable)`

Tabla 2. Ejemplos de operaciones en API funcional

El uso del broadcast no supone un gran impacto para los dispositivos de menor capacidad, ya que, por la arquitectura de e-balance el mayor impacto de las comunicaciones recae en los dispositivos de mayor capacidad como PCs o servidores y, por otra parte, la arquitectura fractal aisla en diferentes niveles el impacto.

5.5 Despliegue del sistema

Tal y como se ha comentado, e-balance utiliza varias MUs para la gestión de los diferentes niveles de la red. Son estas MUs las que intercambian información utilizando el middleware descrito en las secciones anteriores. Las aplicaciones, utilizan a la plataforma de comunicaciones con el objetivo de lograr un uso más eficiente de la red eléctrica. En esta sección se describe parte de dicho despliegue en tres niveles: CMU, LVGMU y MVGMU.

- CMU: En cada casa se dispone de una CMU, que controla los electrodomésticos inteligentes y monitoriza el consumo/generación de energía de la casa. Cada CMU

requiere conexión a Internet, utilizando la conexión de la casa donde se instala. Las CMUs se ejecutan sobre dispositivos BeagleBone Black [4]. Estos dispositivos son de bajo coste y escaso tamaño. Están equipados con 512 MB de DDR3L DRAM, 4GB 8-bit eMMC de flash memory, procesador AM3358 a 1GHz basado en el procesador ARM Cortex-A8, aceleradora 3D, y un acelerador NEON para punto flotante. La BeagleBone está equipada con USB, Ethernet y Micro HDMI. La conexión Wi-Fi puede realizarse a través de un *dongle* utilizando el Puerto USB.
El software se ejecuta en la BeagleBone utilizando Mono [5], una implementación de código abierto de Microsoft .NET. Adicionalmente, la CMU ejecuta aplicaciones Java, que utilizan la plataforma de comunicaciones para acceder a otras MUs.

- LVGMU y MVGMU: Este tipo de MUs ejecutan también el mismo software que las BeagleBones. La implementación actual incluye el uso de máquinas virtuales y restricciones adicionales en las comunicaciones, que por motivos de simplicidad no son incluidas en este artículo.

6 Conclusiones

El desarrollo de un sistema de gestión de redes electricas inteligentes comprende una gran variedad de dispositivos hardware y elementos software que hacen muy difícil la reutilización de código. Con el fin de facilitar el desarrollo de aplicaciones en este tipo de plataformas, en este artículo se ha presentado un middleware centrado en datos que puede ser utilizado en los diferentes niveles de la arquitectura para comunicación y almacenamiento de datos en tiempo real. El middleware dispone de una interfaz de datos simple y una funcional, lo que facilita su reutilización y adaptación en diferentes sistemas. Además, la interfaz de datos está basada en servicios web RESTful, lo que permite su utilización desde prácticamente cualquier tipo de dispositivo y sistema operativo. Por último, el middleware tiene que encargarse además de los aspectos relativos a tiempo real y seguridad. Actualmente se está utilizando en el marco del proyecto europeo e-balance para el despliegue de sus demostradores.

7 Referencias

1. Página web proyecto e-balance. http://www.e-balance-project.eu/
2. K. Piotrowski, et al., “Deliverable D3.2 – Detailed System Architecture Specification”, Public deliverable of e-balance project, FP7-Smartcities-2013, Project number 609132, 2015.
3. ServiceStack framework. https://servicestack.net/
4. Beaglebone Black. http://www.beagleboard.org
5. Mono. Cross platform, open source .NET framework. http://www.mono-project.com

Agradecimientos

Este trabajo ha sido parcialmente soportado por el proyecto europeo e-balance (609132), nacional POLYCIMS (TIN2014-52034-R) y regional MIsTIca (TIC-1572).

Comportamiento del middleware de comunicaciones Ice en entornos con y sin virtualización

Jorge Domínguez Poblete, Marisol García Valls, Christian Calva Urrego

Universidad Carlos III de Madrid
Av. de la universidad 30, 28911 Leganes, Spain
mvalls@it.uc3m.es

Resumen El middleware de comunicaciones tuvo su origen ya hace décadas, tiempos aquellos en los que el hardware existente (y la capacidad de las redes de comunicaciones) era sustancialmente diferente a las posibilidades de estas tecnologías en el momento presente. Concretamente, a medida que se ha introducido tecnologías de virtualización que permiten utilizar de forma más efectiva los servidores, se ha ido observando efectos a estudiar sobre los retardos de comunicación entre nodos. En este artículo se analiza el comportamiento de un middleware de comunicaciones concreto, Ice (Internet Communication Engine) que es una evolución de Corba. Se observan efectos interesantes y difíciles de explicar sobre todo para hardware con una capacidad de cómputo relativamente pequeña, en comparación a los grandes servidores típicos de centros de procesamiento en entornos de computación en la nube.

Keywords: middleware, prestaciones, software de virtualización, Internet Communications Engine (Ice)

1. Introducción

Los sistemas distribuidos actuales se ejecutan sobre nodos y redes cuyo hardware ha sufrido una evolución vertiginosa, sobre todo en la última década. La inundación del mercado con procesadores de gran capacidad de cómputo y de almacenamiento, la aparición de redes de comunicaciones de anchos de banda del orden de 100Gb, el avance de las tecnologías de virtualización con el consiguiente aumento en la consolidación de servidores [8], ha creado un entorno de ejecución radicalmente diferente al que encontró el middleware en sus orígenes.

En ciertas áreas de aplicación, como las relativas a los sistemas de tiempo real críticos, la utilización de middleware no está recomendada sobre todo en aquellas partes de nivel de criticidad mayor. Ello es debido a que tradicionalmente el middleware cuenta con una serie de librerías que utilizan técnicas de programación sofisticada más orientada a ofrecer un entorno de programación amigable, de gran productividad de programación, y comunicación fiable, eficiente y rápida en entornos de propósito general que a ofrecer garantías temporales estrictas.

Este es el caso para la gran mayoría de las tecnologías existentes como Corba [4], Java RMI [3], Ice [2], JMS [6], implementaciones de REST [7], AMQP [11], Storm [14], River [13], or JBoss [15], entre otros. Incluso el estándar OMG más popular en el momento como DDS [5] ofrece parámetros de calidad de servicio pero en ningún caso garantiza cotas máximas en los tiempos de comunicación. También Ada DSA [12] ofrece una extensión del lenguaje ada para sistemas distribuidos de tiempo real aunque está restringido al propio lenguaje.

Los sistemas distribuidos y de tiempo real y su evolución a los sistemas ciber-físicos requieren que los nuevos diseños de middleware de tiempo real tengan unas características específicas [18] que incluyan la verificación on-line de nuevas configuraciones [17]. Dentro de los sistemas distribuidos de tiempo real se han realizado varias contribuciones al diseño e implementación de middleware de tiempo real pero en su mayoría son aspectos que solucionan partes concretas de la predecibilidad en la comunicación. Por ejemplo, [9] es un middleware para sistemas distribuidos de tiempo real que requieren reconfiguración dinámica basado en gestión de calidad de servicio en los nodos [10] y algoritmos de reconfiguración [19]. [20] presenta un diseño inicial de middleware de tiempo real centrado en el ajuste de dos parámetros básicos y [21] presenta una arquitectura distribuida centralizada que soporta un número dinámico de clientes conectados a un servidor a través del manejo de su threadpool.

En este trabajo se analiza el comportamiento temporal de un middleware de comunicaciones de propósito general que surgió como una evolución de Corba. Este middleware es orientado a objetos, implementado en C++ y ofrece diferentes modelos de interacción, principalmente, basado en invocaciones remotas (RPC), con su consiguiente evolución a los objetos distribuidos, y en publicación-suscripción (P/S).

El artículo presenta la estructura que sigue. En la sección 2 se describe la arquitectura de Ice. En la sección 3 se analiza el comportamiento de Ice para un entorno sencillo (de ejecución directa sobre el hardware) y para un entorno virtualizado sobre KVM. La sección 4 presenta las conclusiones del trabajo y las líneas futuras y de continuación.

2. El middleware Ice

El diseño de Ice fue influenciado de forma clara por Corba pero con el objetivo de evitar incorporar los mismos errores de éste. Sin embargo, el API es bastante simple y muy fácil de usar. Su modelo de programación es potente ya que los objetos remotos pueden implementar diferentes interfaces remotas bajo una misma entidad de objeto. Permite utilizar comportamientos dinámicos enviando código de acceso al cliente y activando servidores bajo demanda para evitar un consumo de recursos cuando no es necesario.

2.1. Arquitectura y características

Ice es un framework de código abierto que soporta que máquinas o procesos cliente puedan comunicar entre sí estando implementados en múltiples (y dife-

rentes) lenguajes de programación como Java, C++, C#, PHP, Python, Ruby, y algunos más, y que puede ejecutar sobre varios sistemas operativos como Linux, Windows, OS X, Android, Solaris e iOS. Ice es un middleware de comunicaciones orientado a objectos que permite la creación de aplicaciones distribuidas cuya comunicación se basa en un protocolo de invocación a procedimiento remoto (RPC). El objetivo de los protocolos RPC, típicamente usados en comunicaciones del tipo cliente-servidor, es el de ejecutar funciones remotas (es decir, ubicadas en una máquina servidora, distinta de la máquina que invoca o cliente) de forma síncrona, devolviendo un resultado que puede ser nulo `void`.

Los middleware basados en RPC pueden utilizar diferentes lenguajes de programación en el servidor y en el cliente. Por ello, se establece un acuerdo explícito de las funciones remotas disponibles en el servidor para clientes a través de un *lenguaje de definición de interfaz* o IDL (*Interface Definition Language*) que en Ice se denomina *Slice*.

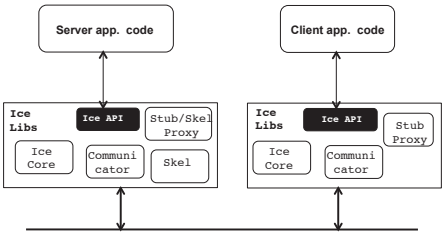


Figura 1. Vista esquemática de la arquitectura de Ice

La figura 1 muestra la arquitectura básica de Ice. Una aplicación o sistema distribuido con Ice contiene al menos una parte cliente y una parte servidora. Tanto cliente como servidor contienen código de nivel de aplicación y las librerías y run-time de Ice. Ice genera además código a partir de la definición de la interfaz Slice.

La parte indicada como **Ice Core** es el núcleo principal de la lógica del middleware que contiene el soporte del run-time para la comunicación remota tanto del lado cliente como del lado servidor. El *core* contiene, en su mayoría, los detalles de la red, concurrencia, ordenación de bytes y otros aspectos relacionados con la red que es aconsejable mantener alejados del código de aplicación.

Existe una parte del *core* que es dependiente de los tipos específicos declarados en Slice y utilizados en la comunicación entre cliente y servidor y otra parte que es independiente que se denomina **generic part of Ice core**. La parte genérica es accedida a través de el **Ice API**, que es idéntica tanto para clientes

como servidores y se encarga de tareas básicas de la administración de Ice como inicialización y finalización del run-time, entre otras tareas.

Para la gestión remota de la invocación, tanto en el lado del cliente como en el lado del servidor, aparecen unas entidades adicionales que reciben el nombre de **proxy**, concretamente son *stub* y *skeleton*. El código de los proxies se crea a partir de la definición de la interfaz IDL y, por lo tanto, son específicos de los objetos y datos que contiene la interfaz. *Stub* y *skeleton* son entidades que posibilitan (acercan) la comunicación con (y desde) los objetos remotos. Son, por tanto, representantes del objeto o código remoto que posibilitan que el cliente realice la traducción de las invocaciones a un formato neutro para su transmisión y el servidor pueda reconstruir estas invocaciones y las envíe hasta el nivel de aplicación como si se tratara de una invocación local.

El *stub* ofrece: (*i*) una interfaz para invocar los métodos de una interfaz de un objeto remoto; y (*ii*) código para *marshalling* y *unmarshalling*. Marshalling es el proceso de serializar o aplanar una estructura compleja de comunicación y convertirla en un formato que pueda ser transmitido por la red. El código de marshalling convierte una invocación a una secuencia de transmisión estándar que es independiente de las reglas de *endianness* y *padding* de la máquina local. El proceso de unmarshalling es lo contrario y deserializa los datos que llegan por la red, reconstruyéndolos a la representación local y tipos de datos apropiados para el lenguage de programación que se use. El *skeleton* es el código equivalente en el lado servidor, que ofrece una interfaz de llamada (up-call) para que el run-time de Ice transfiera el hilo de control al código de aplicación. También contiene el código de marshalling y unmarshalling de forma que el servidor puede recibir parámetros enviados por el cliente y devolver parámetros y excepciones al cliente.

2.2. Objetos específicos

El objeto *Communicator* o comunicador es el punto de entrada de todas las interacciones (es decir, invocaciones). Despacha las invocaciones a todas las facilidades y entidades de las librerías de Ice y controla el *thread pool*.

Un objeto *servant* o sirviente tiene la responsabilidad de gestionar y soportar las ejecuciones de las operaciones remotas del lado servidor ya que las invocaciones solicitadas se realizan en un lenguaje de programación específico.

Otra entidad importante de Ice es el objeto *adaptador*, que es específico del lado servidor. Su función es la siguiente [1]:

- Hace corresponder una invocación entrante al método específico de objetos de lenguage de programación. Por tanto, conoce qué sirvientes están cargados en la memoria y en qué objetos.
- Un adaptador tiene uno o varios puntos de entrada (*transport endpoint*). Si es el caso que un adaptador tenga varios transport endpoints, los sirvientes del adaptador podrán, a su vez, ser alcanzados vía múltiples transportes. Esto permite asociar dos *transport endpoints* a un mismo adaptador para ofrecer diferentes niveles de rendimiento y calidad de servicio.

- Es responsable de la creación de proxies que pueden pasarse a los clientes. El adaptador conoce el tipo, identidad y características del transporte de cada uno de sus objetos e include la información adecuada cuando el código de la parte servidora pide la creación de un proxy.

2.3. Seguridad

Una característica importante de Ice es que ofrece transmisión segura a través del protocolo SSL (*secure socket layer*), que es un estándar de facto para las comunicaciones en red seguras. Aunque en aplicaciones de propósito general el rendimiento experimentado no es excesivo, sí que debe ser considerado en entornos con restricciones de tiempo ya que combina una serie de técnicas criptográficas como: encriptación de clave pública, encriptación simétrica (de clave compartida), códigos de autenticación de mensajes y certificados digitales. Todo ello se traduce en que al establecer una conexión vía SSL se realiza un protocolo de *handshake* durante el cual, se validan los certificados digitales que identifican a los participantes en la comunicación y se intercambian las claves simétricas para encriptar el tráfico de la sesión. Debido a lo pesado de la utilización de encriptación de clave pública, ésta sólo se usa durante el espacio de handshake y cuando éste acaba, SSL utiliza códigos de autenticación de mensajes para asegurar la integridad de los datos. Esto permite que cliente y servidor comuniquen con garantías razonables de que sus mensajes están seguros.

3. Comportamiento temporal de Ice

En esta sección se describen las pruebas de rendimiento realizadas un sistema distribuido con Ice tanto con máquinas sin virtualizar como con máquinas virtualizadas.

En las pruebas *sin virtualizar* se utilizó un despliegue en dos máquinas Intel Celeron 3400 de doble núcleo a 2.6 GHz y 2GB y el sistema operativo Linux distribución Ubuntu 12.04.5 LTS de 32 bits y gcc v 4.6.3. Ice 3.4 y compilador gcc v 4.6.3.

Para el *entorno virtualizado* se utilizaron las mismas máquinas físicas y QUEMU Virtual CPU version 2.0.0 de 2 núcleos y 1GB RAM. Además se utilizó KVM en su versión asociada al kernel 3.19.0-49-generic #55 14.04.1-Ubuntu SMP. Se utilizó Ice v 3.5 y gcc version 4.8.4.

3.1. Código de ejemplo

En esta sección se muestra el código de un cliente que obtiene un proxy de un servidor y posteriormente realiza dos invocaciones remotas sobre dos objetos remotos. Este código tiene validez para todos los despliegues, tanto virtualizados como sin virtualizar.


```
ic=Ice::initialize(argc,argv);
Ice::ObjectPrx base= ic->stringToProxy("CService:default -h
localhost -p 10000");
CServicePrx remoteService=CServicePrx::checkedCast(base);
Ice::ObjectPrx base1 = ic->stringToProxy("CommService:default -
    -h
localhost -p 10001");
CommServicePrx remoteService1 = CommServicePrx::checkedCast(
    base1);
if (!remoteService)
throw "Invalid proxy";
remoteService -> server_op1();
remoteService1 -> server_op2();
```

La inicialización del run-time se realiza mediante un objeto *communicator* (`ic`) que permite acceder al núcleo mediante la función `stringToProxy` para obtener un proxy del objeto remoto con nombre público `CSService`. En este caso el objeto reside en la misma máquina local y en el puerto indicado (`10001`). Para invocar las operaciones del objeto remoto (`server_op1` y `server_op2`) se utiliza TCP, que se indica por defecto.

A continuación se muestra el código del servidor en el que se crean dos objetos remotos, cada uno en un adaptador diferente.

```
ic=Ice::initialize(argc,argv);
Ice::ObjectAdapterPtr adapter1 = ic->
 createObjectAdapterWithEndpoints("adapterA", "default -p
     10000");
Ice::ObjectPtr object1 = new CServiceI;
adapter1->add(object1, ic->stringToIdentity("CService"));
adapter1->activate();

Ice::ObjectAdapterPtr adapter2 = ic->
 createObjectAdapterWithEndpoints("adapterB", "default -p
     10001");
Ice::ObjectPtr object1 = new CommServiceI;
adapter1->add(object1, ic->stringToIdentity("CommService"));
adapter1->activate();

ic->waitForShutdown();
```

En la parte servidora se crea un objeto adaptador que permite que el servidor pueda publicar diferentes objetos servidores con distintas interfaces (`CServiceI` y `CommService`).

3.2. Resultados

En esta sección se muestran los resultados obtenidos para el sistema distribuido con Ice para los siguientes escenarios con el cliente y servidor en la:

- misma máquina física
- diferente máquina física
- misma máquina virtual
- diferente máquina virtual dentro de la misma máquina física
- diferente máquina virtual en diferente máquina física

En todos los casos se ha obtenido 10,000 medidas de invocaciones, mostrándose la media.

En la figura 2 se muestra el rendimiento en un escenario sin carga tanto en las máquinas físicas como en la red. Se puede derivar un comportamiento que corresponde, en general, al esperado, aunque no para todos los casos. A mayor tamaño de datos enviados, mayor es el tiempo que se tarda en procesar y transmitir la información. No existe una diferencia destacable entre los escenarios en los que la aplicación ejecuta en la misma máquina física o en la misma máquina virtual.

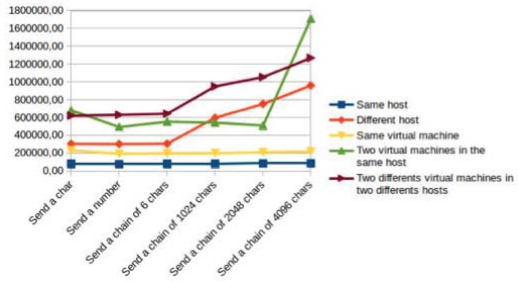


Figura 2. Rendimiento para un escenario sin carga

También se puede decir que el comportamiento es mejor de forma sostenida para el caso en que tenemos dos máquinas físicas sin virtualizar respecto al caso en el que se comunica dentro de la misma máquina física estando en cliente y servidor en máquinas virtuales diferentes.

En la figura 3 me muestra el rendimiento para un escenario con carga elevada, concretamente se ejecuta la parte servidora en una máquina con un 98 % de carga de forma sostenida. En este caso se observa que los escenarios de distinta máquina física y de dos máquinas virtuales en distintas máquinas físicas acusan esta saturación del sistema de forma más evidente.

Se observa también que los casos misma máquina física y misma máquina virtual tiene tiempos prácticamente iguales aunque existe un caso en el que el tiempo de comunicación supera al resto de casos previsiblemente peores.

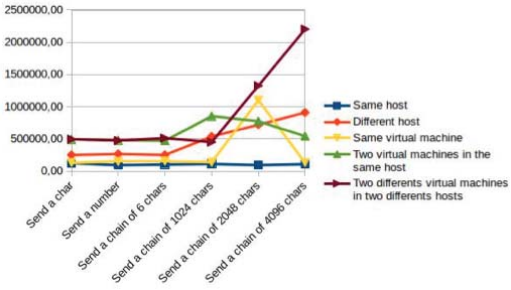


Figura 3. Rendimiento para un escenario con carga

En general, las máquinas utilizadas no poseen un hardware muy potente y, al no haberse utilizado ningún tipo de optimización para la virtualización, ello también influye en el comportamiento observado.

4. Conclusión

En este trabajo se ha presentado el middleware de comunicaciones Ice que ofrece un rendimiento excelente para aplicaciones distribuidas con requisitos de tiempo no estrictos. Se ha realizado una presentación de las características de este middleware y un análisis de su comportamiento temporal en entornos distribuidos tanto virtualizados como no virtualizados. Se observa que Ice presenta tiempos muy estables para la comunicación entre cliente y servidor en la misma máquina (con si sin virtualización). Sin embargo, en cuanto la distribución es real, el sistema acusa sobre todo la existencia de comunicación entre dos máquinas virtuales diferentes, siendo el rendimiento peor el observado en máquinas físicas diferentes que a su vez están virtualizadas. En presencia de carga elevada, el middleware no consigue mantener los tiempos de invocación en los valores normales y presenta mayor inestabilidad. Por tanto, se deduce que para cargas por debajo de 90 %, Ice presenta un comportamiento estable, ofreciendo tiempos de invocación sostenidos.

Agradecimientos

This work has been partly funded by the project REM4VSS (TIN2011-28339) and M2C2 funded by the Spanish Ministry of Economy and Competitiveness.

Referencias

1. ZeroC. Documentation for Ice 3.6. Client and Server Structure. https://doc.zeroc.com/display/Ice36/Client+and+Server+Structure (on-line) (2016)
2. ZeroC Inc.: The Internet Communications Engine. http://www.zeroc.com/ice.html (2003)
3. Sun Microsystems: Java^{TM} Remote Method Invocation API http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/ (on-line) (2016)
4. Object Management Group: The Common Object Request Broker. Architecture and Specification, Version 3.3 (November 2012) http://www.omg.org/spec/CORBA/3.3
5. Object Management Group.: A Data Distribution Service for Real-time Systems Version 1.2. Real-Time Systems. (2007)
6. Deakin, N.: JSR 343: Java^{TM} Message Service 2.0. Oracle. (2013)
7. Richardson, L., Ruby, S.: RESTful web service. O'Reilley Media. (2011)
8. García-Valls, M., Cucinotta, T., Lu, C.: Challenges in real-time virtualization and predictable cloud computing. Journal of Systems Architecture, 60(2): 726–740. (2014)
9. García-Valls, M.,Fernández-Villar, L., Rodríguez-López, I.: iLAND An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems. IEEE Transactions on Industrial Informatics, 9(1): 228–236. (2013)
10. García-Valls, M., Alonso, A., and de la Puente, J.A.: *A Dual-Band Priority Assignment Algorithm for QoS Resource Management*. Future Generation Computer Systems, 28(6): 902–912. (2012)
11. ISO/IEC Information Technology Task Force (ITTF). *OASIS AMQP1.0 – Advanced Message Queuing Protocol (AMQP), v1.0.* ISO/IEC 19464:2014. (2014)
12. Ada Core.: PolyORB. Ada Distributed Systems Annex (DSA). http://www.adacore.com/polyorb (on-line) (2016)
13. Apache Software Foundation. *Jini^{TM} network technologies specifiation. Apache River v2.2.0.* (https://river.apache.org/doc/spec-index.html) (2013)
14. Apache Software Foundation. *Storm 0.10.0.* http://storm.apache.org (on-line) (2015)
15. JBoss. *JBoss Messaging*. http://docs.jboss.org (on-line) (2015)
16. Microsoft. *Disstributed Component Object Model (DCOM)*. (on-line) (2016)
17. Bersani, M. M., García-Valls, M.: The Cost of Formal Verification in Adaptive CPS. An Example of a Virtualized Server Node. 17th IEEE International Symposium on High Assurance Systems Engineering (HASE 2016), pp. 39-46. Orlando, FL. (2016)
18. García Valls, M., Baldoni, R.: Adaptive middleware design for CPS: Considerations on the OS, resource managers, and the network run-time. Proc. 14th Workshop on Adaptive and Reflective Middleware (ARM@Middleware). (2015)
19. García-Valls, M., Uriol-Resuela, P., Ibánez-Vázquez, F.: Low complexity reconfiguration for data-intensive service-oriented applications. Future Generation Computer Systems, vol.37. (2014)
20. García-Valls, M., Calva-Urrego, C., Alonso, A., de la Puente, J. A.: Adjusting middleware knobs to suit CPS domains. 31st ACM/SIGAPP Symposium on Applied Computing. Pisa, Italy. (2016)
21. García-Valls, M.: A proposal for cost-effective server usage in CPS in the presence of dynamic client requests. 19th IEEE Symposium on Real-time computing and distributed applications (ISORC). York, UK. (2016)

Análisis de herramientas de generación automática de código para modelos Simulink *

Beatriz Lacruz, Jorge Garrido, Juan Zamorano y Juan A. de la Puente

Grupo de Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos (STRAST)
Universidad Politécnica de Madrid
str@dit.upm.es

Resumen

Las técnicas de desarrollo basado en modelos persiguen facilitar el desarrollo de sistemas en varios aspectos. Entre ellos, el diseño basado en modelos permite a expertos de diversos campos diseñar e implementar sistemas sin necesidad de conocimientos avanzados de programación. Esto es posible gracias al creciente número de herramientas que permiten generar código funcional a partir de modelos. Una de las herramientas más extendidas para el diseño de sistemas de control entre ingenieros de diferentes ramas de conocimiento es Simulink. En este artículo analizamos las capacidades y rendimiento de dos herramientas de generación automática de código para modelos Simulink. Este análsis se motiva e ilustra con su aplicación al proceso de desarrollo del control de actitud del satélite universitario UPMSat-2.

1. Introducción

El satélite UPMSat-2 es un micro-satélite experimental desarrollado dentro del ámbito de la Universidad Politécnica de Madrid como demostrador tecnológico. En el ámbito de este proyecto, liderado por el instituto Ignacio Da Riva (IDR)[1], participan tanto grupos de investigación de la UPM como diferentes empresas y organismos del sector aeroespacial tanto a nivel nacional como europeo.

La participación del grupo de Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos (STRAST)[2] comprende el desarrollo del software tanto del

*Este trabajo ha saido parcialmente financiado por el Plan Nacional de I+D+i del Ministerio de Economía y Competitividad (projecto M2C2, TIN2014-56158-C4-3-P).

[1] www.idr.upm.es

[2] www.dit.upm.es/str

segmento de vuelo como del segmento de tierra, así como parte del desarrollo e implementación del hardware asociado a ambos segmentos.

El software de a bordo se está desarrollando en el lenguaje de programación Ada, aplicando las restricciones al lenguaje para su uso en sistemas de alta integridad definidas en el perfil de Ravenscar [3]. El computador de a bordo donde ejecutará este código está basado en una síntesis del procesador LEON3 [6]. La compilación del código para dicha plataforma de ejecución se realiza con el conjunto de herramientas GNATforLEON [9] incluyendo el kernel de tiempo real Open Ravenscar Kernel (ORK) [5] también desarrollado por el grupo STRAST.

Uno de los subsistemas principales dentro del software de a bordo del UPMSat-2 es el Attitude Determination and Control System (ADCS). Este subsistema se encarga de mantener la orientación del satélite con respecto a la Tierra. Dicho subsistema se ha implementado mediante un sistema de control desarrollado por ingenieros del grupo IDR usando la herramienta de diseño basada en modelos Simulink[3].

En este artículo presentamos una comparación entre las herramientas de generación automática de código disponibles para Simulink a propósito de la experiencia en su uso para el UPMSat-2. Estas herramientas de generación de código utilizadas han sido la herramienta propia incluida en Simulink así como la herramienta QGen de AdaCore[4].

Las herramientas de generación automática de código resultan de gran utilidad en el desarrollo de sistemas multidisciplinares, al facilitar la interacción entre ingenieros de distinto ámbito de conocimiento gracias al desarrollo basado en modelos. Sin embargo, el uso de estas herramientas afectan al conjunto del ciclo de vida del software. Por tanto, no solo se requiere que el código generado sea correcto funcionalmente, si no que debe estar alineado con el resto del software en cuanto a los requisitos no funcionales [2]. Entre los más relevantes se encuentran la certificabilidad del código generado, cumplimiento de parámetros calidad y estilo que faciliten su mantenimiento, así como su analizabilidad temporal. En este artículo, por tanto, el análisis se centra en estos aspectos concretos que han determinado el grado de idoneidad de ambas herramientras para su uso en sistemas de tiempo real y en el UPMSat-2 en particular.

El resto del artículo se estructura de la siguiente manera: en la sección 2 se presenta más en detalle el diseño del subsistema (ADCS), en la sección 3 se introduce el proceso de generación automática de código, mientras que en la sección 4 se realizan diferentes comparaciones del cógido generado por las herramientas evaluadas. Finalmente, en la sección 5 se ofrecen las conclusiones del trabajo realizado.

2. Sistema de control de actitud

El sistema de control de actitud (ADCS) del satélite UMSat-2 tiene como objetivo el control de la orientación del satélite respecto del eje de referencia

[3] *Simulink* es una marca registrada de The MathWorks Inc.
[4] `www.adacore.com/qgen`

de la Tierra. Concretamente, esta actitud u orientación se controla con respecto a los tres ejes mostrados en la figura 1. Tanto la toma de datos como la actuación para el algoritmo de control empleado se realizan en base al campo magnético de la tierra. Los sensores principales del sistema son un conjunto de magnetómetros (dos nominales y uno experimental) cada uno monitorizando los tres mencionados ejes. Los actuadores del sistema son un conjunto de magnetopares que, generando un campo magnético propio, son capaces de hacer interactuar este con el propio campo magnético terrestre con el fin de producir la actitud deseada.

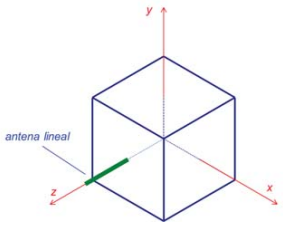


Figura 1: Ejes y antena del satélite UPMSat-2.

Dentro de la fase de operación del subsistema a bordo del satélite se pueden distinguir tres fases principales. La primera, de estabilización inicial, consiste en reducir la alta velocidad de giro producida por la separación del vehículo lanzador hasta alcanzar la actitud nominal. La segunda es la fase de operación nominal en la cual el sistema se encarga de mantener la actitud comandada. Finalmente, existe una fase experimental, en la cual se realizará un extenso programa de validación tanto del funcionamiento del algoritmo de control bajo diversas configuraciones, como de los diferentes dispositvos de toma de datos y actuación experimentales que conforman parte de la carga de pago del satélite.

En general, la actitud nominal deseada del satélite es la mostrada en la figura 2, donde los ejes X e Y no presentan rotación alguna, mientras que se produce una ligera rotación sobre el eje Z, siendo esta rotación de 0,1 radianes por segundo. Esta actitud tiene dos objetivos principales. El primero es la correcta alineación de la antena del satélite con respecto a la superficie de la Tierra. Dado que la antena del satélite será un dipolo omnidireccional, este ha de situarse paralelo a la superficie terrestre para maximizar la potencia recibida desde la estación de tierra. El segundo objetivo de la actitud comandada al ADCS es contribuir en el control térmico, así como al correcto funcionamiento y durabilidad de los paneles solares al repartir homogeneamente la exposición solar de la superficie del satélite gracias a la rotación sobre el eje Z.

Esta actitud nominal puede verse alterada por diversos factores. De entre ellos, el más influyente es el campo magnético terrestre, y sus fluctuaciones sobre la superficie en función de la latitud. Otras fuentes de alteración del campo

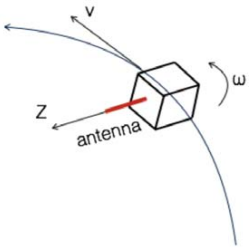


Figura 2: La actitud programada para el satélite incluye una lenta rotación sobre el eje Z.

magnético terrestre son la variante radiación solar a lo largo de la órbita, así como la resistencia provocada por la atmósfera residual existente a la altitud de vuelo, de unos 700km de altura.

Para obtener y mantener la actitud deseada, los ingenieros aeronáuticos del grupo IDR han desarrollado un algoritmo de control [4] basado en una variante de la ley de control *B-dot*. Este algoritmo ha sido diseñado mediante la herramienta de diseño basado en modelos Simulink. Así mismo, este modelo incluye la simulación de las dinámicas del satélite, aplicando tanto las actuaciones realizadas por el algoritmo de control como los efecto de las perturbaciones mencionadas anteriormente. De este modo, se puede realizar una simulación completa de la evolución de la actitud del satélite gracias a la realimentación del resultado de cada ciclo del algoritmo de control, como muestra la figura 3.

Para la implementación del algoritmo de control en el software de a bordo del satélite UPMSat-2 se ha seguido una estrategia basada en la generación automática de código a partir del modelo de Simulink. Concretamente se ha generado el código funcional del algoritmo de control (bloque *controller*), que ha sido integrado en el esquema de tareas del código Ada aplicando las restricciones del perfil de Ravenscar. De este modo se facilita el desarrollo, validación y verificación en términos de análisis temporal. En la sección 3 se detallan el proceso y las herramientas utilizadas para este cometido.

3. Generación automática de código

El aumento del uso de entornos basados en modelos ha impulsado la generación automática de código y con ello la aparición de nuevas herramientas que permiten generar código partiendo del modelo.

La generación automática de código constituye una fase importante en este tipo de sistemas ya que permite obtener código consistente con el modelo diseñado. Una de las mayores ventajas que se obtienen con el uso de estas

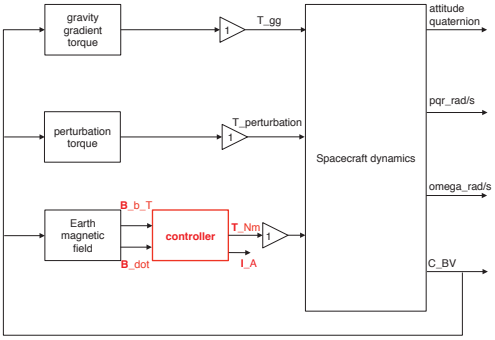


Figura 3: Esquema general del modelo Simulink.

herramientas es la disminución e incluso la eliminación de errores de carácter humano introducidos durante la implementación del código.

Existen diferentes herramientas que posibilitan la generación automática de código a partir de modelos, la herramienta Matlab/Simulink incluye un generador automático de código propio, para modelos diseñados en Matlab existen otros generadores de código como TargetLink® de dSpace o QGen de Adacore. A la hora de elegir entre unas herramientas u otras los principales puntos de decisión son:

- **Certificabilidad.** Dependiendo de la herramienta utilizada, el código generado podrá ser o no certificado. Además, algunas herramientas incluyen funcionalidades específicas para facilitar este cometido.
- **Lenguaje de programación.** Cada herramienta de generación de código permite la generación para unos u otros lenguajes de programación. Se deberá verificar la compatibilidad de la herramienta seleccionada con los requisitos del sistema.
- **Optimización.** Las herramientas de generación automática de código ofrecen diferentes opciones de optimización, algunas de ellas dependientes de la plataforma de ejecución.

Dentro del proyecto UPMSat-2 y para el Attitude Determination and Control System (ADCS) del satélite se han utilizado tanto el generador de código integrado en la herramienta Matlab/Simulink sobre la que está diseñado el modelo del ADCS como el generador de código QGen desarrollado por AdaCore.

En concreto, se ha generado código para el bloque especificado en la sección 2. Este código se ha integrado en el código Ada utilizado para la implementación

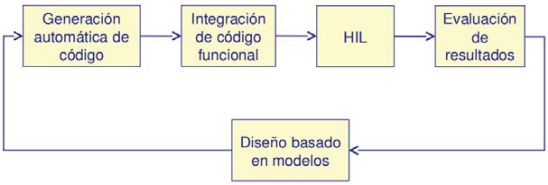


Figura 4: Ciclo de desarrollo basado en modelos realimentado.

del sistema. Las actividades de verificación y validación se han llevado a cabo mediante procedimientos de Hardware In the Loop (HIL) [7].

3.1. Matlab/Simulink

Simulink es una herramienta que forma parte de Matlab y que permite crear diseños basados en modelos para la generación automática de código y simulación.

Matlab incluye un generador automático de código integrado en Simulink que permite generar código a partir del modelo de una forma rápida. Cuenta con múltiples opciones para la generación del código pudiendo generar código en lenguaje C, C++, Verilog® o VHDL®.

La posibilidad de seleccionar el tipo de hardware sobre el que se va a ejecutar el código así como la gran cantidad de opciones disponibles para la optimización del código hacen que sea un generador muy completo. Sin embargo, su principal contrapartida es que el código generado no es certificable.

3.2. QGen

QGen es la herramienta de generación automática de código desarrollada por AdaCore. Entre otros, QGen puede generar código para modelos generados con Simulink. Esto puede hacerse mediante línea de comandos o integrando la herramienta QGen en el entorno gráfico de Simulink.

A diferencia de Simulink, QGen permite generar código en lenguaje C o en Ada/SPARK [1] ofreciendo la ventaja de que éste puede ser certificado. Debido a esto, QGen únicamente soporta la generación de código para el subconjunto de bloques Simulink de los cuales es posible generar un código con las características requeridas en sistemas de alta integridad.

Al igual que el generador de Simulink, QGen ofrece diferentes opciones para una generación de código configurable en términos de optimización y legibilidad.

QGen se ha utilizado como segunda herramienta para generar código debido a que permite la generación de código en lenguaje Ada, que es el lenguaje de

programación utilizado en el proyecto UPMSat-2 y por tanto la integración con el código concurrente resultaba más sencilla.

4. Comparación

Partiendo del modelo del control de actitud se ha generado código con las herramientas mencionadas en la sección 3, pudiendo realizar las siguientes comparaciones entre el código generado por ambas herramientas:

4.1. Optimización

Las opciones para optimizar el código son diferentes dependiendo de la herramienta utilizada. Cada herramienta ofrece diferentes opciones a elegir antes de la generación del código.

En el caso de Simulink, al generar código C, se debe limitar las opciones de optimización de la herramienta para evitar el uso de características del lenguaje incompatibles con sistemas de alta integridad. Este es el caso por ejemplo de la función *memset*, que es la función que Simulink utiliza por defecto para la inicialización de valores. En la figura 5 pueden verse las opciones de optimización ofrecidas por el generador de Matlab/Simulink y que se han seleccionado para generar el código C.

Para generar código Ada mediante QGen no se ha seleccionado ninguna opción de optimización, únicamente se ha seleccionado la opción correspondiente a que al generar código en Ada no detecte errores ni warnings correspondientes a MISRA C [8]. El código C generado con QGen se ha obtenido con las mismas opciones. En la figura 6 se pueden ver las opciones de optimización ofrecidas por QGen, así como las utilizadas en la generación del código analizado.

4.2. Legibilidad

La legibilidad es una de las características en la que más diferencia podemos encontrar al comparar ambas herramientas.

En primer lugar se debe tener en consideración que el código se ha generado en dos lenguajes diferentes (C y Ada) y que esto tiene una alta influencia en la legibilidad del código. Por lo general, el código implementado en lenguaje C es menos legible que el código implementado en lenguaje Ada. A esto hay que añadir que el código C generado por el generador automático de la herramienta Simulink es muy poco legible.

Esto es debido principalmente a que la nomenclatura utilizada para variables auxiliares no es descriptiva, utilizando nombres como “tmp1” o “unnamed-id”, mientras que el generador QGen nombra estas variables de forma consistente con la nomenclatura del modelo. Realizando una comparación entre el código C generado mediante Simunlink y el código C generado mediante QGen llegamos a la misma conclusión, ya que la nomenclatura seguida por QGen para generar C produce un código más legible.

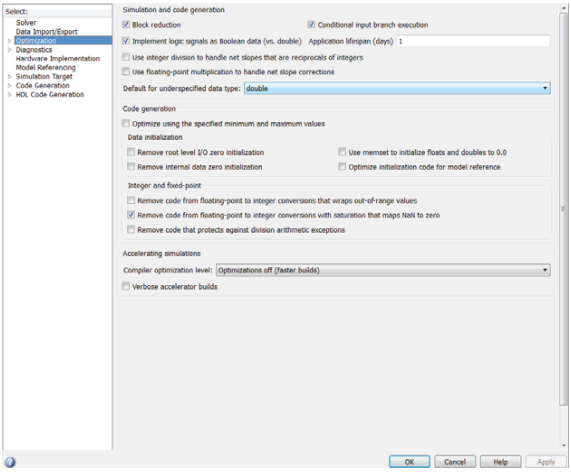


Figura 5: Opciones Simulink

Otra característica que diferencia la legibilidad de ambos códigos es el tratamiento de matrices. Mientras el código generado con QGen (tanto en C como en Ada) las representa mediante vectores bidimensionales, Simulink utiliza una representación basada en vectores unidimensionales.

Finalmente, cabe destacar la diferencia entre el número de ficheros generados por ambas herramientas así como la proporción entre código útil y comentarios. Se ha obtenido que QGen genera un menor número de ficheros así como menor número de líneas de código totales. Además, la documentación se basa en comentarios sobre el propio código teniendo una proporción de aproximadamente una línea de comentarios por cada dos líneas de código útil. En el caso de Simulink, la documentación no solo se basa en comentarios sobre el código sino que también genera documentación HTML, con trazabilidad al propio modelo. En general, la documentación generada por la herramienta Simulink es mucho más completa y significativa que la generada por QGen.

4.3. Tiempo de ejecución

Para evaluar el rendimiento del código generado, se ha realizado un análisis del tiempo de ejecución de ambas versiones de dicho código. Este procedimiento se ha llevado a cabo siguiendo el enfoque descrito en [7].

Para realizar una comparación entre ambos código en tiempo de ejecución

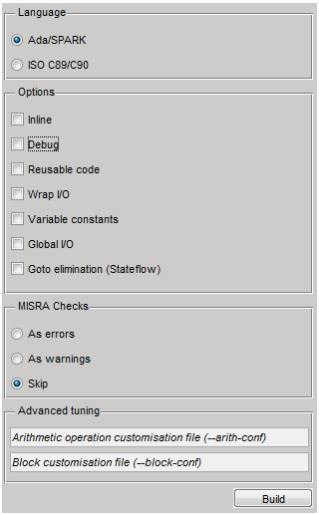


Figura 6: Opciones QGen

se ha utilizado el código generado para una parte del control de actitud, en concreto para el código generado del mismo bloque de simulink el resultado de este análisis muestra que el peor tiempo de ejecución en el caso del código C es de 0.5237 milisegundos, mientras que el tiempo de ejecución en el caso del código Ada es de 0.6860 milisegundos.

Este tiempo se ha obtenido realizando medidas a las llamadas desde el código concurrente a la funciones generadas en código C por Simulink y a los procedimientos obtenidos en caso del código generado en Ada por QGen.

Por tanto, obtenemos que generando código de un mismo bloque del modelo con ambas herramientas el código C generado con el generador automático de Matlab es más rápido que el código Ada generado a través de QGen.

5. Conclusiones

El proyecto UPMSat-2 está sirviendo como demostrador de multitud de tecnologías y procedimientos tanto para grupos de investigación de la UPM como para las empresas e instituciones participantes. En particular, el subsistema de control actitud (ADCS) está teniendo un especial interés, dada la estrecha

interacción requerida con los ingenieros aeronáuticos implicados en este subsistema. Durante el desarrollo de dicho subsistema, se ha podido constatar la facilidad que ofrece el desarrollo basado en modelos para sistemas interdisciplinares, ofreciendo un medio de comunicación técnico independiente del ámbito de conocimiento.

En este trabajo hemos explorado y analizado otra de las notables ventajas del desarrollo basado en modelos, como es el creciente número herramientas que permiten la generación automática de código funcional a partir de dichos modelos. En concreto, se ha generado código funcional a partir de un modelo Simulink haciendo uso de tanto la herramienta propia de Simulink como del generador QGen de AdaCore.

Si bien la generación automática de código ofrece diversas ventajas, también implica algunas contrapartidas, sobre todo en términos de mantenimiento, verificación y validación. Entre ellas, se encuentran la pérdida de control de la calidad del código, que ya no depende del equipo de desarrollo si no de la herramienta de generación automática. En este sentido, se ha realizado un análsis del código producido por las herramientas utilizadas, focalizado en la legibilidad, optimización y rendimiento del código generado.

Se ha observado que el código generado mediante Simulink tiene una baja legibilidad en comparación con la del código generado con QGen. Además, al generar código con Simulink el conjunto de opciones de optimización seleccionado debe de ser compatible con la plataforma utilizada.

No obstante, en el caso de uso presentado el código generado en C con Simulink ha ofrecido de forma consistente menores tiempos de ejecución que el código Ada generado por QGen. Dada la sensibilidad de este sistema en concreto a los tiempos de respuesta, este parámetro es de especial relevancia.

Por el contrario, la no certificabilidad del código generado por Simulink resulta un obstáculo insalvable en determinados ámbitos, como pueden ser sistemas de alta integridad. En este aspecto, la herramienta QGen de AdaCore ofrece una solución apropiada para dichos sistemas.

Como consecuencia de todo lo anterior, se puede concluir que en función de la aplicación en cuestión y plataforma de despliegue objetivo, así como los requisitos no funcionales del sistema, será recomendable el uso de una herramienta u otra.

Referencias

[1] John Barnes. *High Integrity Ada. The SPARK Approach.* Addison Wesley, 1997.

[2] Matteo Bordin and Tullio Vardanega. Automated model-based generation of Ravenscar-compliant source code. In *Proc. 17th Euromicro Conference on Real-Time System*, ECRTS '05, pages 59–67, Washington, DC, USA, 2005. IEEE Computer Society.

[3] Alan Burns, Brian Dobbing, and Tullio Vardanega. Guide for the use of the Ada Ravenscar profile in high integrity systems. *Ada Letters*, XXIV:1–74, June 2004.

[4] Javier Cubas, Assal Farrahi, and Santiago Pindado. Magnetic attitude control for satellites in polar or sun- synchronous orbits. *Journal of Guidance, Control, and Dynamics*, pages 1–12, aug 2015.

[5] Juan A. de la Puente, José F. Ruiz, and Juan Zamorano. An open Ravenscar real-time kernel for GNAT. In Hubert B. Keller and Erhard Plödereder, editors, *Reliable Software Technologies — Ada-Europe 2000*, number 1845 in LNCS, pages 5–15. Springer-Verlag, 2000.

[6] Gaisler. *LEON3 - High-performance SPARC V8 32-bit Processor. GRLIB IP Core User's Manual*. Gaisler Research, 2012.

[7] Jorge Garrido, Daniel Brosnan, Juan A. de la Puente, Alejandro Alonso, and Juan Zamorano. Analysis of WCET in an experimental satellite software development. In Tullio Vardanega, editor, *12th International Workshop on Worst-Case Execution Time Analysis*, volume 23 of *OpenAccess Series in Informatics (OASIcs)*, pages 81–90. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012.

[8] The Motor Industry Software Reliability Association. *MISRA-C:2004. Guidelines for the use of the C language in critical systems*, 2004.

[9] José F. Ruiz. GNAT Pro for on-board mission-critical space applications. In Tullio Vardanega and Andy Wellings, editors, *Reliable Software Technologies — Ada-Europe 2005*, volume 3555 of *LNCS*. Springer-Verlag, 2005.

Sistemas operativos y gestión de recursos

Selección de una arquitectura *many-core* comercial como plataforma de tiempo real

David García Villaescusa, Michael González Harbour y Mario Aldea Rivas

Dpto. Ingeniería Informática y Electrónica
Universidad de Cantabria, Santander (España)
{garciavd, mgh, aldeam}@unican.es

Resumen Los procesadores *many-core* representan la evolución natural de las arquitecturas de computadores de propósito general. Su aumento de prestaciones y la contribución a la disminución de tamaño, peso y consumo del sistema completo con respecto a los procesadores actuales hace prever que serán usados también como plataformas para sistemas de tiempo real en el futuro. En este artículo se identifican los requisitos que debería cumplir una arquitectura *many-core* para su utilización en sistemas de tiempo real y se analizan varios procesadores *many-core* existentes en el mercado para acabar seleccionando el mejor candidato. Finalmente el artículo adelanta algunos de los retos esperables en el desarrollo del soporte para aplicaciones de tiempo real sobre estos sistemas.[1]

Palabras clave: Tiempo real · Many-core · Mapeado · Planificación · Sistemas operativos · Intel Xeon Phi · Tilera · Kalray

1. Introducción

Históricamente los procesadores mejoraron su potencia de cómputo a base de subir la velocidad de su reloj interno. Esto elevaba el consumo implicando un límite al crecimiento impuesto por la disipación del calor generado por los procesadores. Como solución, desde hace ya varios años se está orientado la evolución de los procesadores al aumento del número de núcleos de cómputo. En la actualidad, los procesadores *multi-core* son un referente en la electrónica de consumo. Esto es debido a las múltiples ventajas que proporcionan respecto a un *mono-core*:

- Mejor rendimiento de manera más eficiente.
- Permiten la ejecución simultánea de múltiples aplicaciones en un solo procesador, reduciendo el consumo y las altas exigencias de refrigeración con respecto a los procesadores con muy altas frecuencia de reloj. En sistemas embebidos de altas prestaciones esto permite una reducción del número de sistemas de computación, que equivale a una reducción de peso y cableado.

[1] Este trabajo ha sido financiado en parte por el Gobierno de España en el proyecto TIN2014-56158-C4-2-P (M2C2).

- Permiten que se efectúen diferentes tareas de una misma aplicación de manera simultánea utilizando diversos hilos de ejecución con concurrencia física. Esto supone un aumento en la capacidad de cómputo, pero requiere de un esfuerzo a la hora de paralelizar el código.

Sin embargo, el uso de un bus para el acceso a la memoria compartida y el mantenimiento de la coherencia de cachés introduce incertidumbre en los tiempo de acceso a la memoria y con ello en el tiempo de ejecución de peor caso, lo que lleva a desaconsejar en general la implantación de *multi-cores* en sistemas de seguridad crítica [1][4][10]. Estos sistemas requieren de un proceso de certificación para asegurar que son capaces de cumplir con ciertos criterios de seguridad y fiabilidad. Existen sectores, como la aviación, que poseen sus procesos de certificación específicos.

La utilización de un único bus, compartido por todos los núcleos, para el acceso a la memoria constituye el principal cuello de botella en las arquitecturas *multi-core*. Dicho bus compartido es la principal causa de la baja escalabilidad de este tipo de arquitecturas que ha conducido a la búsqueda de nuevas estrategias de acceso como las que se emplean en los sistemas *many-core* en los que el número de núcleos puede aumentar gracias a un cambio en la arquitectura de comunicación con la memoria. Estas arquitecturas son novedosas y apenas hay experiencia de su uso en sistemas de tiempo real. En este artículo pretendemos analizar las arquitecturas de algunos sistemas *many-core* comerciales para estudiar cuál de ellos se puede adaptar mejor a los requisitos de un sistema de tiempo real en el que es preciso tener garantías de predictibilidad de los tiempos de respuesta.

La estructura del artículo consta de una exposición de los requisitos generales que ha de cumplir una plataforma de tiempo real seguida de un análisis de tres procesadores *many-core* comerciales: Intel, Tilera y Kalray. Sobre ellos se ha realizado un estudio del cumplimiento de los requisitos planteados, ubicado en la sección 2. A continuación, en la sección 3 se describen aspectos generales sobre el mapeado de hilos de ejecución a núcleos y en la sección 4 se hace un breve resumen de algunos sistemas operativos disponibles para arquitecturas *many-core*. Finalmente, en la sección 5 se expone la conclusión de qué procesador es el candidato con mayor potencial para llevar a cabo un estudio más completo de la implementación de sistemas de tiempo real en procesadores *many-core*.

2. Procesadores *many-core* como plataformas de tiempo real

Las principales diferencias entre procesadores *multi-core* y *many-core* son el número de núcleos y la forma en la que estos se conectan. Las arquitecturas *many-core* utilizan una arquitectura más escalable que el bus, tal como una malla o un toro. De esta manera se mantienen, incluso se mejoran, las ventajas que supone el uso de múltiples núcleos a la vez que se aumenta la potencia de cómputo.

Como hemos comentado, la actual generación de procesadores *multi-core* no es adecuada para ejecutar software de tiempo real a su máxima potencia, debido a la impredecibilidad en el tiempo de acceso a memoria. Se recomienda su uso solo si se cumplen determinadas restricciones, por ejemplo una de las más drásticas consiste en que la aplicación se ejecute solo en un núcleo [1]. Por ello es de gran interés poder incluir la próxima generación de procesadores, los *many-core*, como plataformas para sistemas de tiempo real y explorar la posibilidad de hacerlo incluso en sistemas de seguridad crítica. Es preciso investigar si los procesadores *many-core* poseen una arquitectura más predecible en cuanto a los tiempos de respuesta.

2.1. Requisitos generales

Para poder utilizar procesadores *many-core* en sistemas de tiempo real hay que cumplir ciertos requisitos en las especificaciones del mismo:

1. **Núcleos predecibles.** Los núcleos deben proporcionar de manera individual un comportamiento acotable temporalmente. Un aspecto que puede incidir negativamente en la predictibilidad es la ejecución fuera de orden *out of order*. Esto afecta a los tiempos de ejecución e incide negativamente en los cambios de contexto en los que de manera asíncrona el procesador abandona un punto de la ejecución de un hilo para pasar a ejecutar otro hilo, debiendo por tanto descartar cálculos hechos fuera de orden. Este retraso se puede añadir con facilidad al tiempo de cambio de contexto. Pero por otro lado la ejecución fuera de orden presenta el inconveniente de dificultar el análisis estático de tiempos de ejecución.
2. **Memoria local.** La existencia de memoria compartida en un procesador genera una gran incertidumbre en el cálculo de los tiempos de respuesta. En muchos sistemas *multi-core* tanto el último nivel de memoria caché (L2 o L3) como el bus de memoria son compartidos. Además, la memoria caché privada (L1) necesita mantener la coherencia de sus datos, lo cual provoca que determinados accesos a datos contenidos en esta caché no sean independientes si se hacen accesos a ellos desde diferentes núcleos. La coherencia de las memorias caché genera una gran incertidumbre en el tiempo de respuesta máximo salvo que se tengan mecanismos de coherencia de caché con tiempo de ejecución acotada o se pueda deshabilitar dicha coherencia de caché. Para sistemas de tiempo real, se necesitan cachés privadas o memorias locales a los núcleos de tamaño suficiente para albergar la mayor parte de los accesos de un thread o hilo de ejecución alojado en cada núcleo.
3. **Mapeado de memoria.** La existencia de memorias locales o cachés privadas hace necesario disponer de mecanismos para mapear la memoria usada por un hilo de ejecución y bloquearla en la caché privada a voluntad.
4. **Red de interconexión.** La Network-on-Chip (NoC) también tiene que proporcionar un comportamiento temporal predecible. No basta con que tenga un gran *throughput* (anchura de banda) y baja latencia.
5. **Periféricos.** Diferentes aplicaciones pueden intentar acceder a un mismo periférico. El acceso debe de ser controlado y sincronizado.

6. **Acceso a memoria.** Se requieren tiempos de acceso a memoria acotados.
7. **Envío de mensajes.** Es preciso poder de acotar el tiempo de envío de mensajes entre los diferentes núcleos del procesador. Puesto que en muchas redes el envío de mensajes a procesadores vecinos es más rápido que a otros más lejanos la posibilidad de controlar el mapeado de hilos de ejecución a procesadores puede facilitar el agrupamiento de aquellos hilos que tienen mayores relaciones entre sí.

Vamos a analizar a continuación tres arquitecturas de procesadores comerciales disponibles actualmente y estudiaremos el cumplimiento de los requisitos expuestos arriba (con las limitaciones de la documentación pública existente).

2.2. Intel Xeon Phi

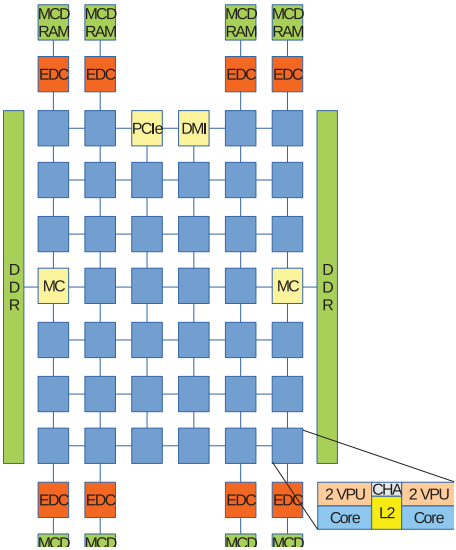


Figura 1. Arquitectura del procesador Intel Xeon Phi Knight's Landing (izq.) y arquitectura de un núcleo del procesador (der.)

La Intel Xeon Phi es una familia de procesadores que lleva evolucionando desde 2010 y cuya generación actual se llama Knights Landing (KNL) [15]. Esta arquitectura está orientada a la computación de altas prestaciones. Como se puede observar en la figura 1 su arquitectura consiste en una malla 2D que conecta celdas, cada una con 2 núcleos y cada uno con 2 Vector Processing Units (VPUs) que comparten 1MB de caché L2. La coherencia de la caché L2 y la conexión con la NoC se realizan a través de la Caching/Home Agent (CHA). El procesador Intel® Xeon Phi™ Processor 7290 ofrece esta arquitectura Knights Landing y tiene 72 núcleos.

Analicemos cómo se adecúa este procesador a los requisitos de tiempo real:

1. **Núcleos predecibles.** Cada núcleo dispone de ejecución fuera de orden (*out-of-order*) y cuatro hilos simultáneos (*hyperthreading*). Son una modificación de los Intel Atom con mayor búfer *out-of-order*.
2. **Memoria local.** Los núcleos solo disponen de memoria privada en el nivel L1, separada en datos e instrucciones ambas de 32kB. La L2 se comparte en las celdas y se mantiene coherente a través de la malla.
3. **Mapeado de memoria.** La documentación pública no especifica si es posible realizar el mapeado de memoria en cachés por medio de la precarga y bloqueo de determinadas líneas.
4. **Red de interconexión.** Conecta las diferentes celdas de núcleos, controladores de memorias, controladores de entrada/salida y demás elementos del procesador. La malla soporta el protocolo de coherencia de caché MESIF (Modified, Exclusive, Shared, Invalid, Forward).
5. **Periféricos.** Dispone de un coprocesador para PCIe (PCI express).
6. **Acceso a memoria.** Los controladores de memoria están situados en las celdas EDC y MC de la figura 1.
7. **Envío de mensajes.** La NoC utiliza enrutado XY estático (cada salto en dirección Y cuesta 1 ciclo y 2 ciclos en dirección X).

Particionado Se puede particionar el conjunto de celdas en 2 o 4 grupos para crear zonas aisladas entre sí para mejorar las prestaciones. Estas particiones se pueden seleccionar desde la BIOS.

Conclusiones La familia Intel Xeon Phi está muy dirigida a un mercado que nada tiene que ver con el tiempo real y eso se nota en que, aparentemente, no se permite la desactivación de la coherencia de caché L2 y que en cada celda se encuentre un sistema de doble núcleo. El pequeño tamaño de la caché privada L1 hace difícil evitar los conflictos debidos a la coherencia de la caché L2.

2.3. Tilera

Como se puede observar en la figura 2, los procesadores de la arquitectura Tilera-GX [16] se dividen en celdas, cada una con un procesador de 64-bits, memoria caché (niveles 1 y 2) y un *switch* que actúa como interfaz con la malla

que comunica todas las celdas y que proporciona coherencia de la caché L2 a todos los núcleos. El procesador Tile GX-100, de esta familia, es un modelo que presenta 100 núcleos.

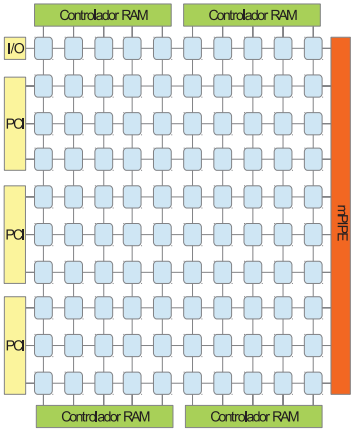


Figura 2. Arquitectura del procesador TILE-Gx100

Vamos a analizar el cumplimiento de los requisitos de tiempo real en este procesador:

1. **Núcleos predecibles.** Los núcleos son homogéneos con una arquitectura *Very Long Instruction Word* (VLIW) de 3 vías con instrucciones de 64-bits y ejecución en orden. Disponen de 32kB de caché L1 de instrucciones y otros tantos de datos. Cada núcleo también tiene 256kB de caché L2.
2. **Memoria local.** El espacio de direcciones físicas compartidas tiene coherencia de caché L2 con todo el *hardware*. Utiliza un modelo de memoria compartida y las lecturas/escrituras de entrada/salida se realizan directamente por la caché.
3. **Mapeado de memoria.** Se mapea cada dirección física a un núcleo propietario, con lo que si esa dirección se cachea es en la caché L2 de ese núcleo donde se guarda el dato. El resto de celdas acceden al dato mediante la malla de comunicaciones. De esta manera se minimizan los accesos a la memoria

RAM y se tiene una visión global de una caché compartida, que hace las veces de una caché L3 virtual.

4. **Red de interconexión.** El número de núcleos varía según el modelo pero siempre se distribuyen en una malla 2D. En la NoC se computa y envía en la misma instrucción. Esta posee 5 redes independientes *full-duplex* (IDN: sistema y entrada salida; MSN: fallos de caché, DMA (Direct Memory Access) y demás memorias; TDN: acceso a memoria entre celdas; UDN: streaming a nivel de usuario y STN: transferencias escalares). Las cachés llevan integrados mecanismos DMA para acceder a/desde la malla. Cada celda tiene un *switch*.
5. **Periféricos.** Como se puede observar en la figura 2 el acceso a los periféricos es a través de los núcleos situados en las filas de la izquierda y la derecha del procesador.
6. **Acceso a memoria.** Las instrucciones y datos son manejados por controladores de caché que proporcionan una interfaz al sistema de memorias. Traducen las direcciones de memoria virtuales a físicas además de proporcionar una vista coherente de la memoria. Cuando se requieren datos que no se encuentran en la caché, el controlador utiliza la NoC para comprobar otras caches y la memoria principal.
   La memoria está compartida de manera global y las transacciones se realizan a través de la malla. El *hardware* proporciona una vista de la memoria de las aplicaciones con coherencia de caché, aunque el valor esté en la caché de otra celda.
   Los datos tienen una celda asignada, donde las otras celdas buscarían el valor del dato en caso de fallo de la caché L2. En caso de no encontrarse ahí, se requiere un acceso a la memoria principal DDR.
7. **Envío de mensajes.** Los datos pueden transferirse punto a punto: celda a memoria, entre celdas y de celdas a entrada/salida. Hay conmutación de paquetes, se utiliza enrutado *wormhole* [13] con control de flujo en las celdas cercanas y enrutado X-Y.
   Se tarda un ciclo en pasar de un *switch* a la entrada de un *switch* vecino. Los *switches* soportan dos arbitrajes: *round robin* y *network priority*.
   - Con *round robin* varias entradas están dirigidas al mismo puerto de salida siguiendo estas normas: cada puerto de entrada tiene la misma prioridad y se procesan todas las peticiones al mismo puerto de salida utilizando política cíclica (*round robin*).
   - Con *network priority* la NoC proporciona prioridades dinámicas a los paquetes. Para las entradas de cada *switch* con la misma prioridad se utiliza *round robin*. Se proporciona un contador de antigüedad al núcleo con el objetivo de evitar la inanición y se impide que los núcleos de los bordes del procesador perjudiquen el acceso a los dispositivos de entrada/salida.

Particionado Se puede particionar el conjunto de celdas en grupos de 1 a 36 celdas para crear grupos zonas aisladas entre sí. Este particionado se establece

por software a nivel de hipervisor. En el particionado también se parte y asigna memoria a cada grupo.

Conclusiones El procesador con ejecución en orden es predecible. Pese a ser un sistema que no está orientado a tiempo real se permite cierta independencia entre zonas del procesador gracias al sistema de particionado. También se tiene una NoC de comportamiento predecible. Hay una cierta noción de memoria local si mediante el particionado se consigue que cada celda acceda solo a los datos de su propia caché L2 y no a los de otras deldas. Esta caché L2 tiene un tamaño de 256kB, que puede ser suficiente para un gran número de aplicaciones empotradas.

2.4. Kalray

Kalray es un fabricante que con su chip *many-core* MPA-256 ha conseguido un procesador expresamente dirigido a sistemas embebidos y de tiempo real. Cuenta con 288 núcleos repartidos en 16 agrupaciones (*clústers*) de cómputo y 4 subsistemas de entrada/salida cada uno con 4 núcleos capaces de acceder a la memoria externa. Como se observa en la parte derecha de la figura 3, cada agrupación contiene 16 núcleos de ejecución más un núcleo que actúa como gestor de recursos. Pueden llegarse a conectar 4 procesadores Kalray MPPA-256 a través de una placa TurboCard3 en caso de requerirse mayor potencia de cálculo.

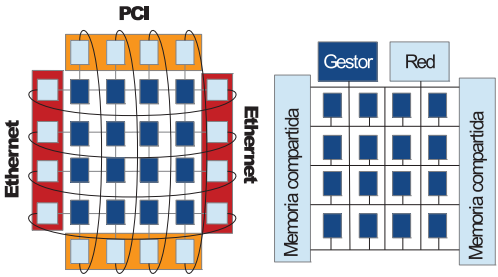


Figura 3. Arquitectura del procesador Kalray MPA-256. A la izquierda el procesador y a la derecha cada agrupación de cómputo

Vamos a analizar el cumplimiento en este procesador de los requisitos de tiempo real:

1. **Núcleos predecibles.** Los 17 núcleos que forman una agrupación de cómputo son idénticos y cada uno posee cachés privadas de 8kB de instrucciones y datos. Cada uno de ellos implementa una arquitectura VLIW de 5 vías con ejecución en orden.
2. **Memoria local.** Cada agrupación tiene una memorial local de 2MB dividida en 16 bancos de 128kB de acceso independiente y sin mecanismo de coherencia. No hay direccionamiento directo de otros espacios de memoria de otros agrupamientos o de la memoria externa. El acceso a cada banco se hace por política cíclica pero es posible configurar la memoria de modo que cada núcleo de ejecución tenga acceso a un banco de memoria diferente y de este modo se evitan todas las interferencias entre ellos.
3. **Mapeado de memoria.** No hay mapeado de memoria directo desde los procesadores de ejecución a la memoria principal o a las memorias locales de los demás agrupamientos.
4. **Red de interconexión.** Tanto las 16 agrupaciones de cómputo como los 4 subsistemas de entrada/salida están conectados por dos NoC de enlaces bidireccionales, una de datos y la otra de control. Ambas presentan una topología toroidal 2D aumentada con conexiones directas entre los subsistemas de I/O. El tráfico de la NoC que pasa por un *router* no interfiere con los buses de memoria, el subsistema de entrada/salida o la agrupación de cómputo salvo que ese nodo sea el destinatario.

   Los canales de datos y control aseguran el envío y los mensajes que siguen la misma ruta llegan en orden. No se envía un mensaje de reconocimiento al nodo fuente. El procesador gestor de recursos tiene conexiones a la interfaz de la NoC a través de líneas de eventos e interrupciones.

   Con objeto de garantizar el cumplimiento de requisitos temporales cada conexión lleva asociada una cuota de anchura de banda, controlada por el nudo emisor.

   Dentro de cada agrupación de cómputo (imagen de la derecha de la figura 3) se utilizan buses que van directamente a los bloques de memoria (divididos en 2 partes, una a la izquierda y otra a la derecha) pudiéndose configurar el acceso a dichos bloques de manera independiente y por tanto con tiempos de acceso acotados.
5. **Periféricos.** El acceso a los periféricos (PCIe y Ethernet) y memoria DDR se realiza a través de los núcleos situados a los bordes del procesador.
6. **Acceso a memoria.** Cada núcleo ve la memoria local a través de cachés L1 de datos e instrucciones con política de reemplazo LRU (Least Recently Used) y sin coherencia hardware, ya que es posible particionar la memoria local entre los núcleos. Es posible implementar mecanismos de coherencia software si fuese necesario compartir algún área de memoria.

   Cada bloque de la memoria compartida del agrupamiento de cómputo es accedido mediante enlaces directos desde los diferentes núcleos mediante arbitrio *round robin*.
7. **Envío de mensajes.** Se puede configurar tanto la ruta y como el flujo para garantizar límites y latencias en el paso de datos desde la fuente. También

se puede configurar el algoritmo de inyección. Van Amstel [3] detalla cómo se garantizan los servicios.
En la temporización hay que considerar las interrupciones entre parejas de núcleos.
La NoC de datos puede operar con garantías debido a los *routers* no bloqueantes y el control de flujo realizado en el origen.

Aplicación a tiempo real Toda agrupación de computo y subsistema de entrada/salida contiene una unidad de soporte de depuración (*Debug Support Unit*) con un contador de 64-bits. Cada contador es direccionable en memoria local y puede leerse por cualquier núcleo. El mensaje de inicialización de este contador, enviado en modo *broadcast*, produce un pequeño desfase entre agrupaciones. Cada núcleo implementa su propio reloj de tiempo real, que da soporte a una implementación ligera de los *timers* POSIX.

Conclusiones Con Kalray nos encontramos con procesadores *many-core* dirigidos a sistemas de tiempo real. La arquitectura anidada nos proporciona un sistema de particionado directo. La NoC tiene tiempos de paso de mensajes acotables y cada núcleo presenta ejecución en orden. Se cuenta con una noción de memoria local, aunque su tamaño de 128kB puede ser algo escaso.

3. Mapeado y análisis

3.1. Mapeado

Para conseguir que un sistema de tiempo real sea ejecutado en procesadores *many-core*, se pueden asignar ciertas tareas de manera permanente a núcleos del procesador, pudiendo llegar a reducir así el tiempo de respuesta de peor caso al minimizar los cambios de contexto. El mapeado de tareas en procesadores *many-core* es un problema NP-Completo por lo que las soluciones utilizadas actualmente son heurísticas. En estos algoritmos se tienen en cuenta la distancia entre los núcleos que se van a comunicar y la congestión que pueda haber en los enlaces que se van a utilizar. Debido a que la complejidad de la colocación en un sistema aumenta de manera exponencial con el tamaño del software (número de tareas) y del *hardware* (cantidad de núcleos), la gran mayoría de algoritmos son heurísticos genéticos [12], simulación [6] o ramificación y poda [9].

También se puede buscar reducir en lo máximo posible el consumo energético utilizando mapeados eficientes de tareas y reduciendo la frecuencia de funcionamiento del procesador mientras se mantiene la planificabilidad [14].

Que hay que tener en cuenta a la hora de mapear tareas en un procesador:

- Las tareas que accedan a dispositivos deben de estar lo más cerca posible a los mismos.
- Las tareas que utilicen asiduamente la memoria RAM deben de colocarse lo más cercanas a su bloque.

- Las tareas que se comuniquen entre ellas deben colocarse cerca.
- Hay que evitar congestiones en la NoC.
- Las tareas más críticas o con menor plazo tienen que estar "mejor" situadas que las demás.

En aplicaciones complejas que demandan flexibilidad puede ser de gran interés poder modificar el mapeado en tiempo de ejecución, en momentos de baja carga del sistema, según las necesidades que tenga el propio sistema.

3.2. Técnicas de análisis

Existen ciertos elementos a modelar, muy dependientes de la arquitectura y de la distancia entre los elementos:

- **Accesos a memoria**: Cuando se accede a memoria local no se compite con otros núcleos. Al acceder a memoria compartida no solo se compite por el propio acceso a la memoria si no por el uso de los enlaces de la NoC. Según la arquitectura, la zona de memoria compartida puede tener bloques reservados a ciertos núcleos, pero sigue siendo necesario analizar la contención en la NoC.
- **Paso de mensajes**: Se compite con el tráfico que utilicen los mismos enlaces que el mensaje. Hay arquitecturas que dividen el tráfico en diferentes *streams* para segregar lo máximo posible el tipo de tráfico.
- **Accesos a dispositivos**: En la mayoría de las arquitecturas los dispositivos están distribuidos a lo largo de la NoC, y por tanto se compite por los enlaces para acceder a ellos.

Los elementos de modelo y las técnicas de análisis que se deriven se podrán añadir en el futuro a una herramienta de análisis de planificabilidad tal como MAST [8], que ya proporciona modelos de sistemas distribuidos.

3.3. Protocolos de sincronización

Los núcleos tienen que ser capaces de sincronizarse a la hora de interactuar entre ellos y cuando utilicen variables compartidas. Es por esto que el sistema operativo debe proporcionar un protocolo de sincronización que suponga poca sobrecarga en la NoC.

3.4. Técnicas de particionado espacial y temporal

Se requieren protocolos con los que mantener las diferentes particiones aisladas de manera espacial y temporal. Las particiones no podrán estar completamente aisladas pues es preciso tener en cuenta que hay cierta contención para el intercambio de mensajes y por el tráfico que atraviesa otras particiones para acceder a recursos compartidos.

En la actualidad el particionado se realiza a la vez que el mapeado de manera estática a través de algoritmos heurísticos. Los algoritmos genéticos se usan

mucho en este contexto y pueden evaluar diferentes características como son: los ciclos que van a tardar en comunicarse diferentes tareas, la probabilidad de que haya comunicación entre las tareas, la contención en los enlaces debida al tráfico y la latencia y probabilidad de acceso a recursos compartidos.

4. Sistemas operativos

Realizar software para sistemas de tiempo real en arquitecturas *many-core* requiere conocimientos de la arquitectura a la hora de diseñar servicios del sistema operativo tales como sincronización de tareas, localización de tareas, localización del código y datos, actualización de datos, acceso a periféricos, etc.

Unos de los elementos más críticos de cualquier sistema operativo es el planificador. Gu realizó un trabajo sobre un planificador de un sistema *many-core* sin memoria compartida [7] proporcionando un protocolo para simplificar la programación paralela y utilizando paso de mensajes entre objetos.

Un sistema operativo de gran interés es Manycore Operating System for Safety-Critical (MOSSCA) [11] que se basa en cumplir los siguientes requisitos y propiedades:

- El sistema debe tener un comportamiento temporal predecible y conducir a la implementación de sistemas analizables.
- El particionado en tiempo y espacio evita interferencias y facilita el análisis. Hay que tener un cuidado especial con los recursos compartidos.
- Los procesadores de la misma aplicación pueden comunicarse entre ellos. Para comunicarse a través de los límites de particionado hay que utilizar el sistema operativo.

MOSSCA es un sistema operativo basado en una arquitectura *microkernel* distribuida en cada núcleo, donde se ofrecen servicios en relación al *hardware* del que dispone cada núcleo.

Otro ejemplo de este tipo de sistemas es eSOL eMCOS [5], que es un sistema operativo de tiempo-real diseñado para embebidos con *many-cores*. Su arquitectura de *microkernel* distribuida incluye servicios básicos tales como paso de mensajes, planificación local y gestión de hilos. Actualmente es compatible con los *many-core* Kalray MPPA2-256 y Tilera Gx8036. La planificación se realiza mediante dos planificadores, uno que asigna las tareas de mayor prioridad a cada núcleo para que puedan ejecutarse siempre que estén listos. El resto de tareas están bajo otro planificador que las distribuye sobre los núcleos balanceando su carga y buscando alto *throughtput*.

5. Conclusión

Se ha seleccionado como mejor contendiente para el reto de llevar el tiempo-real a los procesadores *many-core* la arquitectura de Kalray ya que proporciona características que resultan de gran interés:

- Ejecución en orden dentro de cada núcleo.
- Posibilidad de tener tareas con memoria privada cuando se mapea directamente a un núcleo, disponiendo cada uno de ellos de 128kB de la memoria de la agrupación. El acceso a esta memoria se puede configurar para que no haya interferencias con el resto de núcleos de la agrupación, aunque sí podría haber interferencias con la NoC o el núcleo de gestión.
- No hay coherencia de caché hardware.
- Paso de mensajes para la comunicación entre núcleos con tiempos acotables.
- Sistema anidado escalable.
- Posibilidad de determinar el mapeado de las tareas en el procesador, pudiendo implementar un algoritmo de mapeo.
- Aislamiento entre tareas de diferentes agrupaciones, al menos mientras no necesiten acceder a la NoC para intercambiar mensajes entre sí.

Como se dice en [2] Kalray dispone además de un kit de desarrollo de software junto con un sistema *software* ligero que proporciona un *run-time* capaz de correr software que utiliza servicios POSIX y permite crear sistemas operativos personalizados. También existe un sistema operativo para esta plataforma llamado eSOL eMCOS [5].

Queda como trabajo futuro la adquisición de dicho dispositivo para realizar un estudio más a fondo. Será objeto de este estudio el desarrollo de modelos y técnicas de análisis, heurísticas de mapeado y la implementación de un sistema operativo de tiempo real en el mismo.

Referencias

1. Certification Authorities Software Team: Position Paper-32, Multi-core Processors (2014), http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/media/cast-32.pdf
2. de Dinechin, B.D., Ayrignac, R., Beaucamps, P.E., Couvert, P., Ganne, B., de Massas, P.G., Jacquet, F., Jones, S., Chaisemartin, N.M., Riss, F., Strudel, T.: A clustered manycore processor architecture for embedded and accelerated applications. 2013 IEEE High Performance Extreme Computing Conference (HPEC) pp. 1–6 (2013)
3. de Dinechin, B.D., Durand, Y., van Amstel, D., Ghiti, A.: Guaranteed services of the noc of a manycore processor. In: Proceedings of the 2014 International Workshop on Network on Chip Architectures. pp. 11–16. NoCArc '14, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2685342.2685344
4. EASA: CERTIFICATION MEMORANDUM Subject Development Assurance of Airborne Electronic Hardware (2011), http://www.easa.europa.eu/system/files/dfu/certification-docs-certification-memorandum-EASA-CM-SWCEH-001-Issue-01-Rev-01-Development-Assurance-of-Airborne-Electronic-Hardware.pdf
5. eSOL: emcos, http://www.esol.com/embedded/emcos.html
6. Giannopoulou, G., Stoimenov, N., Huang, P., Thiele, L., de Dinechin, B.D.: Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources. Real-Time Systems 52(4), 399–449 (2016), http://dx.doi.org/10.1007/s11241-015-9227-y

7. Gu, X., Liu, P., Yang, M., Yang, J., Li, C., Yao, Q.: An efficient scheduler of {RTOS} for multi/many-core system. Computers & Electrical Engineering 38(3), 785 – 800 (2012), http://www.sciencedirect.com/science/article/pii/S0045790611001340, the Design and Analysis of Wireless Systems and Emerging Computing Architectures and Systems
8. Harbour, M.G., Gutiérrez, J.J., Medina, J.L., Palencia, J.C., Drake, J.M., Rivas, J.M., Martínez, P.L., Cuevas, C.: Mast: Bringing response-time analysis into real-time systems engineering, http://mast.unican.es/mast_analysis_techniques.pdf
9. Indrusiak, L.S., Harbin, J., Burns, A.: Average and Worst-Case Latency Improvements in Mixed-Criticality Wormhole Networks-on-Chip. 2015 27th Euromicro Conference on Real-Time Systems pp. 47–56 (2015)
10. Jean, X., Berthon, M.G.G., Fumey, M.: MULCORS - Use of Multicore Processors in airborne systems. Journal of Chemical Information and Modeling 53, 160 (1989), https://www.easa.europa.eu/system/files/dfu/CCC_12_006898-REV07-MULCORSFinalReport.pdf
11. Kluge, F., Triquet, B., Rochange, C., Ungerer, T.: Operating systems for manycore processors from the perspective of safety-critical systems. 8th annual workshop on Operating Systems for Embedded Real-Time applications p. 16 (2012)
12. Mesidis, P., Indrusiak, L.S.: Genetic mapping of hard real-time applications onto noc-based mpsocs; a first approach. In: Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on. pp. 1–6 (June 2011)
13. Ni, L.M., McKinley, P.K.: A survey of wormhole routing techniques in direct networks. Computer 26(2), 62–76 (Feb 1993), http://dx.doi.org/10.1109/2.191995
14. Sayuti, M.N.S.M., Indrusiak, L.S.: A function for hard real-time system search-based task mapping optimisation. In: 2015 IEEE 18th International Symposium on Real-Time Distributed Computing. pp. 66–73 (April 2015)
15. Sodani, A., Gramunt, R., Corbal, J., Kim, H.S., Vinod, K., Chinthamani, S., Hutsell, S., Agarwal, R., Liu, Y.C.: Knights landing: Second-generation intel xeon phi product. IEEE Micro 36(2), 34–46 (Mar 2016)
16. Tilera Corporation: Tile Processor Architecture Overview for the Tile-Gx Series (2012), http://www.mellanox.com/repository/solutions/tile-scm/docs/UG130-ArchOverview-TILE-Gx.pdf

Servicios de tiempo real en el sistema operativo Android

Alejandro Pérez Ruiz, Mario Aldea Rivas y Michael González Harbour

Grupo de Ingeniería Software y Tiempo Real, Universidad de Cantabria

{perezruiza,aldeam,mgh}@unican.es

Resumen. Debido a la gran expansión y crecimiento de Android el interés por utilizar este sistema operativo en entornos de tiempo real es cada vez mayor. En este trabajo se describen una serie de mecanismos proporcionados por el sistema operativo Android/Linux mediante los cuales es posible aislar uno o más núcleos de un multiprocesador simétrico para ser utilizados exclusivamente por tareas con requisitos temporales. Gracias a los mecanismos de aislamiento, la tasa de interferencias sufridas por las tareas con requisitos temporales respecto a otras tareas o aplicaciones que se ejecutan en el sistema operativo es muy baja. Un segundo aspecto en el que se mejora el comportamiento de tiempo real del sistema operativo Android está relacionado con las limitaciones para tiempo real de la librería *bionic* (modificación de *glibc* para Android). Para solventar estas limitaciones se ha utilizado la librería *glibc* incluida en la distribución estándar de Linux. Se han realizado una serie de tests que demuestran que la librería tradicional funciona correctamente en Android. Asimismo se ha llevado a cabo la caracterización temporal de Android/glibc para las funciones más relevantes de POSIX para tiempo real observándose que la respuesta temporal del sistema es apropiada para aplicaciones de tiempo real laxo.

Palabras clave: Android, tiempo real, sistemas operativos, multinúcleo

1 Introducción

Desde que apareció el primer móvil con Android este sistema operativo ha estado en constante evolución tanto en características como en soporte para distintas clases de dispositivos. Debido a ello existe un gran interés en utilizar este sistema operativo en entornos que poseen requerimientos temporales. Existen ventajas cuando utilizamos Android para propósitos de tiempo real, ya que es un sistema adaptado y optimizado para utilizar en dispositivos que sean eficientes energéticamente y de recursos limitados, y además es un sistema extendido por un gran número de dispositivos. Lo habitual es que los procesadores modernos usados con dispositivos Android tiendan a utilizar múltiples núcleos.

Este sistema operativo se desarrolla principalmente bajo licencia Apache 2.0, aunque hay determinadas partes que son distribuidas con otros tipos de licencias, por ejemplo, algunos parches del *kernel* tienen licencia GPLv2. Este tipo de licencias de software libre permite a los desarrolladores y a la industria aprovechar las caracterís-

ticas ofrecidas por Android, aplicar parches desarrollados por terceros y realizar modificaciones. Además, el *kernel* de Android está basado en el de Linux. Las últimas versiones de Android que se ejecutan en los dispositivos más modernos utilizan las versiones 3.4 o 3.10 del *kernel* de Linux. Este hecho permite a los desarrolladores utilizar características avanzadas ofrecidas por Linux.

Android está orientado a la ejecución de aplicaciones desarrolladas en Java, aunque es posible ejecutar aplicaciones escritas en cualquier lenguaje utilizando un compilador adecuado. Este sistema operativo proporciona un *framework* que facilita el desarrollo de aplicaciones Java para diferentes dominios. Para alcanzar este objetivo, Android está formado por diferentes capas de componentes software (ver figura 1). La capa más baja corresponde con el *kernel* de Linux, el cual proporciona el funcionamiento básico del sistema, como por ejemplo el manejo de la memoria, los procesos y los drivers. Encima de la capa compuesta por el *kernel*, hay otra capa que contiene un conjunto de librerías nativas que están escritas en C o C++ y son compiladas para una arquitectura hardware específica. Estas librerías incluyen una modificación de la librería C de gnu *glibc* llamada *bionic*. Esta librería está diseñada específicamente para Android pero como analizaremos más adelante presenta algunos inconvenientes cuando es utilizada con aplicaciones de tiempo real.

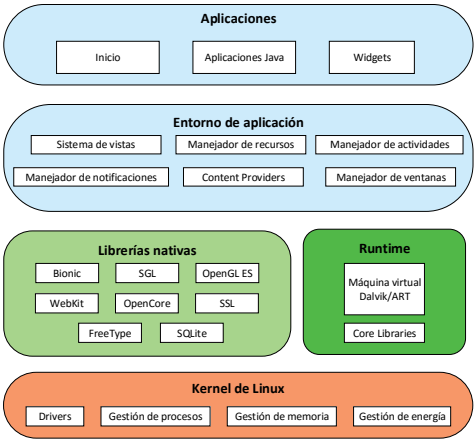


Fig. 1. Arquitectura software de Android

Un componente clave de Android es su máquina virtual para aplicaciones Java. Las últimas versiones de Android incorporan una nueva máquina virtual Java denominada ART (Android RunTime) [1], la cual es la sucesora de la ya obsoleta Dalvik. ART se encarga de ejecutar las aplicaciones escritas en Java compiladas en un formato especí-

fico (*Dalvik executable*) que permiten su portabilidad a través de distintas arquitecturas hardware.

Como ya hemos descrito en un trabajo previo [2], Android al basarse en el *kernel* de Linux ofrece mecanismos que nos permiten aislar un núcleo en un dispositivo con un procesador multinúcleo, de tal modo que se puede utilizar dicho núcleo como un entorno donde ejecutar aplicaciones nativas con requisitos temporales que se ejecutan directamente sobre el *kernel* y las librerías nativas. No obstante, este tipo de aplicaciones utilizan la ya citada librería *Bionic* desarrollada por Google, que no posee todas las características que tiene la implementación tradicional. En particular no soporta ningún protocolo de sincronización para los mutexes, siendo esto un requisito indispensable para cualquier aplicación de tiempo real.

El presente trabajo se estructura del siguiente modo: en la Sección 2 se describen algunos de los trabajos relacionados. En la Sección 3 se muestra la solución que hemos planteado para conseguir ejecutar aplicaciones de tiempo real en Android. La Sección 4 describe brevemente las limitaciones de la librería *Bionic* para aplicaciones de tiempo real. En la Sección 5 se explica cómo superar estas limitaciones utilizando la librería tradicional *glibc* en Android. En la Sección 6 se hace una caracterización de Android para su uso con requisitos de tiempo real. Por último, en la Sección 7 se presentan las conclusiones y los trabajos futuros.

2 Trabajos relacionados

Algunos trabajos [3] [4] [5] [6] han analizado la posibilidad de utilizar Android para ejecutar aplicaciones con requisitos temporales. En ellos se llega a la conclusión de que en primera instancia la plataforma Android es inapropiada para usarse en entornos de tiempo real, a menos que apliquemos mecanismos o modificaciones al sistema operativo. Los principales problemas encontrados en estos estudios son los siguientes:

- La librería *Bionic* tiene más restricciones que la *libc* tradicional y algunas de ellas dificultan su uso en sistemas de tiempo real.
- La variabilidad en los tiempos de respuesta del *kernel* de Android y de la máquina virtual Java impiden tener tiempos de respuesta acotados.
- El *kernel* de Linux utiliza por defecto el planificador llamado "*completamente justo*" (*Completely Fair Scheduler*, en inglés). Este planificador proporciona fracciones temporales dinámicas entre las distintas tareas del sistema para intentar ser lo más justo con todas las tareas que deben ser ejecutadas. Esto es claramente incompatible con los requisitos de predictibilidad de una aplicación de tiempo real.

Debido a las limitaciones citadas anteriormente, un trabajo anterior [5] ha propuesto cuatro posibles soluciones para que Android sea adecuado para ejecutar aplicaciones de tiempo real. En la figura 2a se ilustra la primera solución propuesta, que consiste en ejecutar todas las aplicaciones con requisitos temporales sobre un *kernel* de Linux con características de tiempo real. La siguiente solución que se plantea consiste en añadir una máquina virtual con características de tiempo real (RT- JVM); de este modo gracias a un *kernel* de Linux de tiempo real se podrían ejecutar programas Java

de tiempo real (ver figura 2b). En la figura 2c se ilustra la siguiente propuesta que propone utilizar los servicios de un *kernel* de tiempo real junto con una modificación de la máquina virtual de Android (ART VM extendida). La última solución planteada en la figura 2d utiliza un hipervisor de tiempo real capaz de ejecutar el sistema operativo Android en concurrencia con un sistema operativo de tiempo real.

A raíz de las propuestas anteriores se han llevado a cabo algunos desarrollos para crear extensiones de tiempo real para Android. La solución ilustrada en la figura 2a se ha realizado en un trabajo [7] en el cual se han aplicado una serie de parches (*RT_PREEMPT*) sobre el *kernel* Android/Linux para conseguir características de tiempo real en el sistema. Además para permitir una comunicación entre las aplicaciones Java propias de Android con las de tiempo real han desarrollado un canal de comunicaciones y sincronización entre los dos tipos de aplicaciones.

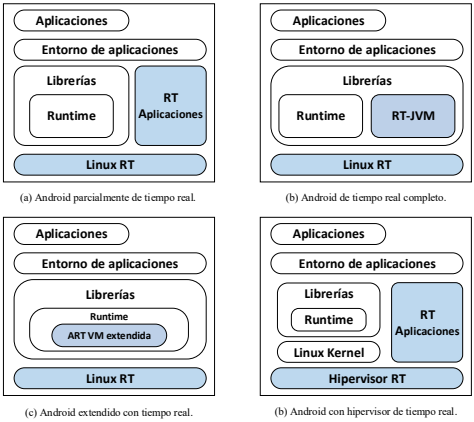


Fig. 2. Soluciones teóricas para la integración de tiempo real en Android [5]. Las partes de color azul indican cambios en la arquitectura de Android.

Otro trabajo [8] [9] ha realizado una adaptación de Android basándose en la solución propuesta en la figura 2c. En este caso también han modificado el *kernel* de Android/Linux con el parche *RT_PREEMPT* y además han añadido modificaciones en otros componentes de Android como en el recolector de basuras de la máquina virtual Java, en la clase *Service* para permitir el cambio de prioridades a nivel de aplicación Java y en el módulo *Binder* para añadir herencia de prioridades en las llamadas remotas a procedimientos dentro de Android.

Existe otro estudio [10] [11] que ha optado por una solución diferente a las cuatro propuestas en la figura 2. En este caso se crea un prototipo denominado RTDroid que pretende ser compatible con las aplicaciones escritas para la plataforma Android. Se

utiliza una máquina virtual de Java con características de tiempo real (Fiji-VM) sobre un sistema operativo de tiempo real (Linux-RT o RTEMS). La API original de Android ha sido recreada paso a paso para poder ejecutar aplicaciones Android. En cierto modo está solución se basa en la figura 2b, pero en este caso, han construido todo el sistema desde cero.

En las dos primeras implementaciones citadas anteriormente nos encontramos con la alta complejidad que reside en la adaptación del *kernel* de Android/Linux para poder aplicar parches de tiempo real sobre él, ya que las últimas versiones del *kernel* de Android no están integradas en la rama principal de desarrollo del *kernel* de Linux. Y la solución propuesta con RTDroid requiere un fuerte proceso de adaptación con cada nueva versión del sistema que aparezca. Debido a esto, en un estudio anterior [2] hemos buscado una solución más portable y fácil de desarrollar y de mantener. En la siguiente sección vamos a describir brevemente dicha solución.

3 Aislamiento de la CPU para ejecutar aplicaciones de tiempo real

Android es un sistema operativo que está en constante evolución y por ello en todas las soluciones citadas en la sección anterior existe una fuerte dependencia con la versión Android utilizada durante su desarrollo. Además, este sistema operativo es una plataforma muy fragmentada [12] lo que hace que todas las soluciones que requieran modificar el *kernel* o la plataforma en general sean difíciles de mantener y extender a distintas clases de dispositivos.

En nuestro estudio previo [2] hemos presentado un mecanismo para ejecutar aplicaciones de tiempo real laxo sobre un dispositivo Android. Para ello las aplicaciones con requisitos temporales (que estarán desarrolladas en lenguaje C) son ejecutadas directamente sobre el *kernel* de Linux en un núcleo aislado de la CPU, sin necesidad de modificar el código del *kernel* ni el de la plataforma. Esta solución puede ser usada en cualquier dispositivo Android con un procesador multinúcleo y *kernel* de Linux versión 2.6 o superior. Dichas aplicaciones con requisitos temporales deben ser ejecutadas haciendo uso de las prioridades de tiempo real que ofrece el *kernel* de Linux/Android (política de planificación *SCHED_FIFO*).

Las versiones del *kernel* de Linux superiores a la 2.6, por defecto tienen activada una opción que hace que la mayor parte de este sea expulsable, por ello en cualquier momento durante la ejecución de código perteneciente al *kernel* se puede producir una expulsión, excepto en los manejadores de interrupciones y regiones protegidas con *spinlocks*. Además en estas versiones del *kernel* también se incluyen políticas de planificación de tiempo real (*SCHED_FIFO* y *SCHED_RR*).

A pesar de todas las características ofrecidas por el *kernel* de Linux/Android existen algunos inconvenientes que debemos solventar si queremos tener un grado de predictibilidad razonable para aplicaciones con requisitos temporales:

- Interferencias entre aplicaciones de tiempo real de diferentes aplicaciones que se pueden estar ejecutando en el sistema.

- Efecto de los manejadores de interrupciones sobre las tareas de la aplicación.
- Cambios dinámicos de frecuencia y apagado automático de los núcleos del procesador.
- Limitaciones de la librería *bionic* para las aplicaciones de tiempo real.

Las tres primeras limitaciones se han resuelto en el estudio previo [2] mientras que en el presente trabajo se resuelve la cuarta limitación relacionada con la librería *bionic*. A continuación se procede a resumir los pasos realizados en el estudio previo [2] para alcanzar nuestra solución que evita interferencias de otras aplicaciones de tiempo real del sistema, elimina interrupciones, fija la frecuencia y evita el apagado de los núcleos del procesador.

Si nos aprovechamos de las ventajas ofrecidas por los procesadores multinúcleo podemos aislar un núcleo de la CPU utilizando mecanismos ofrecidos por Linux/Android para así conseguir un entorno para las aplicaciones de tiempo real. Estas aplicaciones se ejecutarán directamente sobre el *kernel* de Linux/Android y las librerías nativas ofrecidas por el sistema. En la figura 3 se ilustra la arquitectura llevada a cabo para nuestra solución aquí descrita.

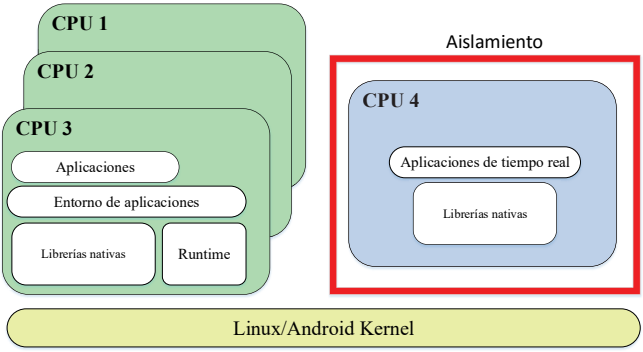


Fig. 3. Solución propuesta en un trabajo previo [2] para ejecutar aplicaciones de tiempo real en Android.

Linux proporciona algunos mecanismos para aislar CPUs de la actividad general del planificador del sistema. El más conveniente para nuestro propósito es el denominado *Cpuset*. Esta funcionalidad proporcionada por el *kernel* no viene activada por defecto en los dispositivos Android, lo que hace que debamos recompilar el *kernel*. Su objetivo consiste en restringir el número de procesadores y recursos de memoria que un

conjunto de procesos pueda utilizar. Cuando el sistema operativo arranca, todos los procesos pertenecen a un único *cpuset*. Si tenemos suficientes privilegios podemos crear nuevos *cpusets* y mover procesos de uno a otro. De este modo podemos mover todos los procesos del sistema a un *cpuset* y al mismo tiempo se puede crear otro *cpuset* donde únicamente se asignen los procesos de tiempo real.

La implementación del mecanismo de los *cpuset* hace que por definición todos los nuevos procesos que se creen se asignen al mismo *cpuset* donde está su proceso padre, a excepción de los procesos hijos de *kthreadd*. Este demonio se ejecuta en el espacio del *kernel* y es utilizado por el sistema para crear nuevos *threads* del *kernel*. Por consiguiente no podemos garantizar que algunos *threads* del *kernel* no se ejecuten en un núcleo aislado. A pesar de ello en los numerosos experimentos realizados esta situación apenas se repite y no tiene gran incidencia sobre los tiempos de ejecución.

Con el anterior mecanismo de aislamiento no conseguimos evitar que lleguen interrupciones a un núcleo aislado, pero a partir de la versión 2.4 del *kernel* de Linux se ha incluido la posibilidad de asignar ciertas interrupciones a un núcleo del procesador (o a un conjunto de núcleos). Esta funcionalidad se denomina *SMP IRQ affinitiy* y nos permite controlar qué núcleos manejan las diferentes interrupciones que se producen en el sistema. Para cada interrupción hay un directorio en el sistema donde existe un fichero que nos permite establecer la afinidad modificando una máscara. No todas las interrupciones son enmascarables, por ejemplo las producidas entre los núcleos del procesador (*IPI-interprocessor interrupts*, en inglés). Al igual que con los *threads* del *kernel* las numerosas pruebas realizadas muestran que el impacto de estas interrupciones es escaso.

Para alcanzar mayor grado de predictibilidad en los tiempos de ejecución es necesario fijar la frecuencia de los núcleos destinados a ejecutar aplicaciones con requisitos temporales. Además, algunos dispositivos Android utilizan demonios que apagan los núcleos de la CPU que no están siendo usados para así poder ahorrar energía. Para evitar que esto ocurra en el núcleo aislado debemos desactivar dichos demonios.

Aplicando todos los mecanismos descritos anteriormente, en el estudio previo que hemos realizado [2] se ha determinado que se consiguen mejoras suficientemente sustanciales en la ejecución de una tarea simple en un núcleo aislado respecto a ejecutar esa misma tarea de una manera no aislada. Esta mejora es suficiente para requisitos de tiempo real laxo.

4 Librería Bionic y sus limitaciones para tiempo real

La implementación de la librería estándar C habitualmente utilizada en Linux es la GNU C Library (comúnmente conocida como *glibc*). Esta librería proporciona soporte para los lenguajes C y C++. Como Android está basado en el *kernel* de Linux parecería lógico asumir que se utilizase esta librería, sin embargo debido a las especificaciones y requisitos de Android se decidió utilizar una nueva librería bautizada con el nombre de *Bionic*. Existen tres razones principales por las que se decidió no utilizar una librería estándar como la *glibc* e implementar una nueva:

- Los dispositivos móviles por norma general tiene un espacio de memoria bastante más limitado que otros dispositivos más "tradicionales", como por ejemplo un ordenador de escritorio.
- Los procesadores que utilizan los dispositivos con Android poseen velocidades inferiores si los comparamos con ordenadores de sobremesa o portátiles.
- Los desarrolladores de Android querían una librería que no estuviese sujeta a las limitaciones de la licencia GPL.

El *kernel* de Linux utiliza una licencia GPL, pero un requisito de Android es que en él se puedan utilizar aplicaciones de terceros que no sólo permitan ser monetizadas sino que en estas aplicaciones se pueda utilizar código propietario. De tal modo que se decidió utilizar una licencia BSD, que sigue siendo de código abierto pero no obliga a que todo el código que se crea a partir de ellas herede la condición de código abierto; por consiguiente los programadores de aplicaciones Android podrán desarrollar código propietario para este sistema operativo.

4.1 Ventajas de la librería *Bionic*

Esta librería tiene optimizado su tamaño debido a que todos los comentarios de los ficheros de cabecera han sido eliminados, y han suprimido todo el código que han considerado inútil para Android. Por otro lado como ya se ha comentado previamente se distribuye con licencia BSD lo que evita cualquier problema legal utilizando software propietario en el espacio de usuario. Por último, existe una ventaja que reside en su optimización para procesadores lentos, para lo que se han realizado cambios en la implementación de la librería *Pthreads* basada en el estándar POSIX. Aunque esto para el caso de querer utilizar esta librería para aplicaciones de tiempo real es un gran inconveniente como veremos a continuación.

4.2 Limitaciones para tiempo real

La librería *Pthreads* [13] que se encuentra en la implementación de la *glibc* tradicional se basa en el estándar POSIX definido por el IEEE para ser compatible a través de diferentes sistemas operativos. Sin embargo como se ha comentado en la subsección anterior la librería *Bionic* ha realizado importantes cambios en este aspecto.

Para determinar si podemos usar *Bionic* en aplicaciones de tiempo real escritas en C hemos ido probando si las funciones más relevantes que se utilizan de manera habitual con requisitos temporales están disponibles. Hemos detectado que algunas tan imprescindibles como las relacionadas con los protocolos de los mutexes no se encuentran implementadas. A continuación se listan las funciones y símbolos no disponibles:

- Mutexes:
  - `pthread_mutexattr_setprotocol`,
  - `pthread_mutexattr_setprioceiling`.
  - `pthread_mutexattr_getprioceiling`.

    - `pthread_mutexattr_setprioceiling`.
- Prioridades:
    - `pthread_setschedprio`.
    - El símbolo `PTHREAD_EXPLICIT_SCHED`.

También existen otras funciones relevantes para aplicaciones de tiempo real que no están implementadas:

- Señales:
    - `sigwaitinfo`.
    - `sigqueue`.
    - `sigtimedwait`.

Estas limitaciones serían más que suficientes para considerar Android un sistema operativo inadecuado para ejecutar aplicaciones de tiempo real, por ello en la siguiente sección se describe la solución adoptada para solventar dichas limitaciones.

5 Glibc en Android

Lo primero que se puede considerar para poder solucionar las limitaciones de tiempo real descritas en la sección anterior podría ser modificar el código de la librería *Bionic* para dar soporte a las funciones no implementadas. Esto supondría un gran esfuerzo y una constante adaptación a las nuevas versiones de la librería que van apareciendo. Por ello, hemos decidido optar por una solución más portable y fácil de aplicar. Esta consiste en utilizar la librería tradicional g*libc* en Android.

Para poder utilizar la librería *glibc* en Android, en nuestro caso[1] es necesario usar una librería compilada y adaptada para arquitecturas ARM que ejecuten un sistema operativo Linux. La forma más directa e inmediata es utilizar un compilador cruzado ARM/Linux con la opción *static* seleccionada, para así conseguir un código ejecutable que incorpore todas las funciones de las librerías de las que haga uso el programa compilado. Esto nos limita a la ejecución exclusiva de programas enlazados estáticamente, de tal modo que también hemos optado por conseguir ejecutar programas que hagan uso de la librería *glibc* en Android de manera dinámica. Para lograr esto se debe realizar lo siguiente:

- Llevar todas las librerías dinámicas al dispositivo Android donde queremos ejecutar las aplicaciones nativas.
- Durante la compilación debemos indicar cuál es el enlazador dinámico y cuál es la ruta donde se encuentran las librerías dinámicas. A modo de ejemplo se expone la orden de compilación para un programa sencillo:

[1] Todas las pruebas se realizan en un Nexus 5 con arquitectura ARM y la versión de Android 6.0.


```
arm-linux-gnueabi-gcc  hello_world.c  -o  hello_world
-Wl,--dynamic-linker=/data/local/libs/ld-linux.so.3
-Wl,-rpath=/data/local/libs -fPIE -pie
```

Teniendo en cuenta que el *kernel* que incorporan los dispositivos con Android no sigue la rama principal de desarrollo del *kernel* de Linux, sería arriesgado afirmar que la librería *glibc* se puede utilizar sin inconveniente alguno en este sistema operativo sin antes realizar algunos tests. Hemos adaptado los test funcionales que están disponibles en el conjunto denominado "Open POSIX Test Suite" [14], donde se realizan pruebas funcionales para *threads*, semáforos, temporizadores, variables condicionales, colas de mensajes y protocolos de herencia de prioridad con mutexes. Todos estos tests han sido pasados satisfactoriamente cuando se ejecutaban en Android haciendo uso de la librería *glibc*, y por consiguiente podemos afirmar que es posible hacer uso de esta librería sobre el *kernel* Linux/Android.

6 Caracterización del sistema para tiempo real

Para poder caracterizar algunas funciones POSIX utilizadas habitualmente con aplicaciones de tiempo real se ha utilizado un teléfono Nexus 5 con un procesador de 4 núcleos a una frecuencia de 2,2 Ghz ejecutando la versión de Android 6.0. Además se han realizado las medidas para dos casos distintos, en el primero de ellos (test A) se miden las funciones en un procesador sin ninguno de sus núcleos aislados; únicamente todos los *threads* de los tests se ejecutan con prioridades de tiempo real. En el otro caso (test B) se ha aislado uno de los núcleos del procesador aplicando los mecanismos descritos en la sección 3. En ambos casos se utiliza la librería tradicional *glibc* y en el sistema se ha establecido una carga de trabajo alta (ejecución de benchmarks y descarga de paquetes por red) para tratar de simular el peor de los escenarios posibles.

	Test A: sin aislamiento	Test B: con un núcleo aislado
Bloqueo de un mutex libre (*lock*)	Mínimo: 1,874 µs Máximo: 1646,301 µs Media: 2,143 µs	Mínimo: 0,261 µs Máximo: 34,064 µs Media: 0,318 µs
Bloqueo de un mutex libre (*trylock*)	Mínimo: 1,874 µs Máximo: 91,303 µs Media: 2,208 µs	Mínimo: 0,261 µs Máximo: 36,304 µs Media: 0,323 µs
Desbloqueo de un mutex (*unlock*)	Mínimo: 1,770 µs Máximo: 111,980 Media: 2,121	Mínimo: 0,261 µs Máximo: 33,387 µs Media: 0,317 µs
Delay de 1000 µs: *clock_nanosleep* (absoluto)	Mínimo: 1030,924 µs Máximo: 5891,718 µs Media: 1112,932 µs	Mínimo: 1027,969 µs Máximo: 1108,230 µs Media: 1067,169 µs
Cambio de prioridad (*pthread_setschedparam*)	Mínimo: 7,970 µs Máximo: 180,053 µs	Mínimo: 7,454 µs Máximo: 64,147 µs

	Media: 15,793 µs	Media: 8,114 µs
Cambio de prioridad (*pthread_setschedprio*)	Mínimo: 8,229 µs Máximo: 130,345 µs Media: 16,162 µs	Mínimo: 7,811 µs Máximo: 68,396 µs Media: 8,446 µs

Tabla 1. Caracterización de algunas funciones POSIX utilizadas habitualmente en aplicaciones de tiempo real.

Los resultados que se muestran en la tabla 1 se han conseguido midiendo la duración de las distintas funciones durante 150000 ejecuciones para cada caso. La tabla 1 arroja una mejora sustancial para el test B (núcleo aislado) donde los tiempos de peor caso son en algunos casos unas 7 veces inferiores y la media de los resultados es alrededor de solo un 15% superior al valor mínimo obtenido en todas las pruebas para el test B (núcleo aislado). Los tiempos máximos observados en el test sin aislamiento pueden ser provocados por cualquier interrupción del sistema o por otras tareas que utilicen prioridades de tiempo real mayores.
Incluso en el caso aislado, en un sistema operativo como Android donde no tenemos pleno control sobre todas las actividades del sistema no se pueden determinar con total exactitud tiempos máximos de respuesta, aunque en los numerosos tests realizados hemos observado que se obtienen valores similares a los mostrados en la tabla 1.

7 Conclusiones y trabajos futuros

En este trabajo se ha presentado una solución para poder utilizar Android con aplicaciones de tiempo real laxo haciendo uso de mecanismos proporcionados por el propio sistema operativo, y aprovechando los procesadores multinúcleo cada vez más presentes en dispositivos Android. Debido a que la librería *bionic* implementada en Android no incorpora todas las funciones utilizadas habitualmente en aplicaciones de tiempo real se ha optado por utilizar la librería tradicional *glibc* usada en los sistemas operativos Linux y se ha testado su correcto funcionamiento en Android. De este modo no es necesaria ninguna modificación del *kernel* ni de ninguna librería proporcionada por la plataforma Android. Esto supone una alta portabilidad para los distintos dispositivos y versiones de Android disponibles actualmente. Algunos tests que hemos realizado sobre la solución propuesta demuestran que se obtienen mejoras sustanciales con respecto al uso de Android sin mecanismos de aislamiento.

Nuestro siguiente objetivo es medir el impacto del uso de drivers de propósito general de Android sobre las aplicaciones de tiempo real que ejecutan en un entorno aislado. Además también es necesario desarrollar un mecanismo de comunicación y sincronización entre las aplicaciones de tiempo real ejecutadas en núcleo aislado y el resto de aplicaciones de Android.

Agradecimientos. Este trabajo ha sido financiado parcialmente por el Gobierno de España con referencia TIN2014-56158-C4-2-P (M2C2) y por el programa de becas predoctorales de la Universidad de Cantabria.

Referencias bibliográficas

1. Andrei Frumusanu (July 1, 2014). "A Closer Look at Android RunTime (ART) in Android L". AnandTech. Retrieved July 5, 2014.
2. Alejandro Pérez Ruiz, Mario Aldea Rivas and Michael González Harbour. “CPU Isolation on the Android OS for running Real-Time Applications”.in Proceedings of the 13th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES), 2015.
3. Bhupinder S. Mongia y Vijak K. Madisetti, “Realiable Real-Time Applications on Android OS”. Whitepaper, 2010.
4. Luc Perneel, Hasan Fayyad-Kazan y Martin Timmerman. “Can Android be used for Real-Time purposes?” in International Conference on Computer Systems and Industrial Informatics, ICCSII ’12, pages 1–6, 2012.
5. C. Maia, L. Nogueira y L. M. Pinho. “Evaulating Android OS for Embedded Real-Time Systems”. En Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, OSPERT 2010, pages 63- 70, Brussels, Belgium, 2010.
6. Luc Perneel, Hasan Fayyad-Kazan, and Martin Timmerman. “Android and Real-Time Applications: Take Care!”, in Journal of Emerging Trends in Computing and Information Sciences, Volume 4, Special Issue ICSSII.
7. W. Mauerer, G. Hillier, J. Sawallisch, S. Hönick, y S. Oberthür. “Real-time android: deterministic ease of use,” en Proceedings of the Embedded Linux Conference Europe (ELCE '12), 2012.
8. Igor Kalkov, Dominik Franke, John F. Schommer, and Stefan Kowalewski. “A real-time extension to the Android platform”. In Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems, JTRES ’12, pages 105–114, New York.
9. Igor Kalkov, Alexandru Gurghian, and Stefan Kowalewski. “Priority Inheritance during Remote Procedure Calls in Real-Time Android using Extended Binder Framework”. ”. In Proceedings of the 13th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES), 2015.
10. Yin Yan, Shaun Cosgrove, Varun Anand, Amit Kulkarni, Sree Harsha Konduri and Steven Y. Ko, Lukasz Ziarek. "Real-Time Android with RTDroid" in Proceedings of the 12th International Conference on Mobile Systems, Applications, and Services (MobiSys), 2014.
11. Yin Yan, Sree Harsha Konduri, Amit Kulkarni, Varun Anand and Steven Y. Ko, Lukasz Ziarek. "RTDroid: A Design for Real-Time Android" in Proceedings of the 11th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES), 2013.
12. Dan Han, Chenlei Zhang, Xiaochao Fan, Abram Hindle, Kenny Wong y Eleni Stroulia. “Understanding Android Fragmentation with Topic Analysis of Vendor-Specific Bugs” in Working Conference on Reverse Engineering (WCRE), 2012.
13. Nichols, Bradford, Dick Buttlar, and Jacqueline Farrell. “Pthreads programming: A POSIX standard for better multiprocessing.” O'Reilly Media, Inc., 1996.
14. Open POSIX Test Suite from A GPL Open Source Project, http://posixtest.sourceforge.net/

Diseño y Gestión de la Demanda Flexible de Recursos en Aplicaciones Multimedia

A. Armentia[1]*, U. Gangoiti[1], E. Estévez[2], M. Marcos[1]

[1] Dept. Ingeniería de Sistemas y Automática. ETSI Bilbao (UPV/EHU), Bilbao, España
{aintzane.armentia, unai.gangoiti, marga.marcos}@ehu.eus
[2] Dept. Ingeniería Electrónica y Automática EPS de Jaén, Jaén, España
eestevez@ujaen.es

Abstract. Muchas aplicaciones multimedia presentan flexibilidad en su demanda de recursos, lo que visto desde el punto de vista del campo de aplicación se traduce en flexibilidad de su QoS. Es decir, soportan cierta degradación de su calidad, lo cual puede resultar muy útil para sobreponerse a situaciones de sobrecarga del sistema. Este trabajo propone una aproximación basada en modelos y multi-agentes para el diseño y gestión de dicha flexibilidad. Más concretamente, se proponen una aproximación de modelado para capturar la flexibilidad de la QoS de nivel de aplicación y su correspondiente mapeo a demanda de recursos. En base a este diseño, un middleware basado en multi-agentes permitirá el mejor nivel de QoS posible para las aplicaciones en ejecución, ajustando su demanda a la disponibilidad de recursos en cada momento.

Keywords: Demanda de recursos flexible · QoS flexible · Sistemas multi-agentes · Diseño basado en modelos

1 Introducción

Las aplicaciones multimedia se pueden emplear en diferentes ámbitos de aplicación con finalidades distintas tales como detección de taras en sistemas de fabricación, detección de incendios o control de acceso en video-vigilancia. A pesar de tener objetivos tan diversos, muchas aplicaciones multimedia comparten características similares. Se trata de aplicaciones que se ejecutan en entornos distribuidos y heterogéneos. Debido a que suelen estar relacionadas con temas de seguridad y privacidad, se debe evitar la interrupción de su servicio incluso en situaciones de caída de nodo, ya que la pérdida de información puede tener efectos no deseados (demandan disponibilidad). Además, estas aplicaciones también comparten otras demandas de flexibilidad.

Por un lado, supervisan su contexto para poder detectar y reaccionar a cambios relevantes en él, demandando, por lo tanto, *adaptabilidad*. Por otro lado, la mayoría presenta *flexibilidad con respecto a su demanda de recursos*, ya que en general soportan cierta degradación de su calidad (QoS del nivel de aplicación) lo que se traduce en una reducción de su demanda de recursos. Por ejemplo, una aplicación que procesa

video de alta calidad para controlar el acceso a un edificio mediante reconocimiento facial, podría proporcionar resultados aceptables usando video de menor resolución. Por lo tanto, una gestión adecuada de esta flexibilidad puede permitir hacer frente a situaciones de sobrecarga en un sistema (arranque de nuevas aplicaciones y/o fallos de nodo), ajustando la demanda de recursos de las aplicaciones en ejecución. Como resultado, la disponibilidad y la escalabilidad del sistema mejoran.

En la literatura varios trabajos tratan las demandas de flexibilidad en aplicaciones distribuidas y dinámicamente adaptables. El paradigma Evento-Condición-Acción parece ser la mejor opción para definir la adaptabilidad a cambios en el contexto en base a la detección de eventos y la ejecución de acciones de respuesta [1]. Por otro lado, los sistemas auto-adaptativos basados en middleware reconfigurables resuelven los problemas de ubicuidad en tiempo de ejecución, siendo también capaces de modificarse a sí mismos a medida que su entorno cambia. Sin embargo, o no son soluciones totalmente genéricas [2] o no dan soporte a aplicaciones con estado [3]. Algo similar ocurre con la disponibilidad del sistema, que o la gestiona la propia aplicación, siendo totalmente consciente del proceso de recuperación llevado a cabo [2], o las aproximaciones propuestas no soportan recuperaciones con estado [3].

En las aplicaciones multimedia la QoS del nivel de aplicación es esencial ya que suele representar la calidad percibida por el usuario. Varios trabajos se han centrado en el diseño de aplicaciones teniendo en cuenta sus requisitos no-funcionales, como en [4] que propone una extensión del leguaje SysML (Systems Modeling Language) para requisitos de rendimiento. Otros trabajos utilizan la definición de la QoS en tareas de composición de aplicaciones, como [5] y [6] en aplicaciones orientadas a servicio. La gestión dinámica de la QoS va un paso más allá, aprovechando la demanda flexible de recursos por parte de las aplicaciones, para adaptarla a la cantidad de recursos disponibles en un determinado instante. Este es el caso del middleware UbiQoS que monitoriza el estado de los recursos del sistema y ajusta los niveles de QoS de las aplicaciones en ejecución cuando es necesario [7]. De nuevo, se trata de una solución en la que el middleware es totalmente consciente de los problemas del dominio de aplicación. Una propuesta más genérica se presenta en [8] que tiene en cuenta el consumo de CPU en cada uno de los posibles niveles de QoS y en [9] donde también se considera el consumo de energía.

En cualquier caso, y hasta donde los autores conocen, no existe ninguna aproximación genérica que combine la caracterización de la flexibilidad en sistemas distribuidos junto con la gestión dinámica de los recursos. Trabajos anteriores de los autores han estado relacionados con el soporte a la flexibilidad en aplicaciones de asistencia domiciliaria [10], estando principalmente orientados a las demandas de adaptabilidad y disponibilidad. En este contexto, el presente artículo completa trabajos anteriores con una definición basada en modelos de la flexibilidad de la QoS de aplicaciones multimedia, mapeándola a su correspondiente demanda de recursos. Además, el middleware basado en multi-agentes se ha extendido con mecanismos de gestión de recursos para poder permitir el mejor nivel de QoS de las aplicaciones en ejecución.

La estructura del artículo es la siguiente: la Sección 2 describe una aproximación de modelado para la captura de la flexibilidad de demanda de recursos en aplicaciones multimedia. También presenta una aplicación de video-vigilancia como prueba de concepto, que será usada para ilustrar el resto de secciones. La Sección 3 propone la arquitectura del middleware basado en multi-agentes (llamado MAS-RECON), resaltando los módulos extendidos. La Sección 4 consta de la evaluación de la propuesta a través del caso de estudio. El artículo termina con las conclusiones y el trabajo futuro.

2 Aproximación de Modelado para Aplicaciones Multimedia Flexibles

En esta sección se identifican los requisitos de aplicaciones multimedia flexibles, y se presenta una aproximación de modelado que cubre dichos requisitos.

Con el objetivo de ilustrar la propuesta, se presenta un caso de estudio para *Control Perimetral*, inspirado en un demostrador del proyecto iLAND [10]. Instalaciones de alto riesgo como los centros penitenciarios suelen disponer de control perimetral: se trata de detectar cuándo un preso intenta escapar, mediante el análisis de la trayectoria de cuerpos en movimiento en señales de video. En el sistema se distinguen dos modos de operación: 1) *Normal*, no hay riesgo de fuga. El objetivo fundamental de este modo es detectar si un objeto, es decir un preso, ha cruzado una determinada línea virtual, momento en el que se debe generar una alarma y pasar al otro modo. Se trata de un modo de operación flexible con respecto a la demanda de recursos, ya que aunque lo deseable sería trabajar con vídeo de alta resolución, se aceptan calidades inferiores sin perder por ello efectividad en el diagnóstico; 2) *Fuga*, en el que se analiza la trayectoria del objeto para lo cual es imprescindible capturar video de gran resolución y alta frecuencia de adquisición. Por lo tanto, no es un modo de operación flexible.

Tal y como se ha comentado anteriormente, los autores introdujeron la aproximación de modelado para el diseño y desarrollo de sistemas distribuidos y flexibles en un trabajo previo, fundamentalmente orientado a los requisitos de adaptabilidad y disponibilidad [10]. El presente trabajo extiende dicha aproximación con mecanismos que permitan cumplir con la flexibilidad en la demanda de recursos. El principal objetivo es poder especificar las aplicaciones de manera que se puedan ejecutar en diferentes niveles de calidad. Para ello, es necesario identificar los niveles de calidad aceptables, la parte de la aplicación afectada por dicha flexibilidad y la cantidad de recursos demandados en cada nivel. Con este propósito, la aproximación incluye nuevos conceptos de modelado para la caracterización de: (1) la QoS de aplicación con diferentes niveles aceptables, por parte del experto de dominio; (2) los recursos demandados en cada nivel, por parte del desarrollador de software. Como resultado, el diseño de la aplicación contendrá toda la información que el middleware necesita para asegurar la mejor calidad posible para las aplicaciones en ejecución, optimizando, al mismo tiempo, los recursos del sistema.

Aunque el trabajo previo de los autores estaba centrado en aplicaciones de asistencia domiciliaria, los siguientes párrafos introducen brevemente los principales conceptos

de modelado (ver Fig. 1), pero con respecto a aplicaciones multimedia. El concepto de Escenario (*Scenario*) agrupa varias aplicaciones que tienen algo en común. Por ejemplo, todas las actividades de vigilancia - concepto de Aplicación (*Application*) – relacionadas con un área concreta de un edificio. Cada aplicación comprende diferentes tareas que pueden ejecutarse en nodos distribuidos y que tienen que cooperar para alcanzar el objetivo de la aplicación – concepto de Componente de Aplicación (*AppComponent*). Teniendo en cuenta que los componentes cooperan mediante el intercambio de los datos necesarios para proporcionar su servicio - concepto de conector de datos (*DataConnector*) - los componentes están provistos de un puerto de entrada (*InputPort*) y/o un puerto de salida (*OutputPort*) para la recepción y envío de datos, respectivamente. Además, dicho intercambio de datos puede estar dirigido por una determinada lógica asociada al puerto de salida (*DataLogic*).

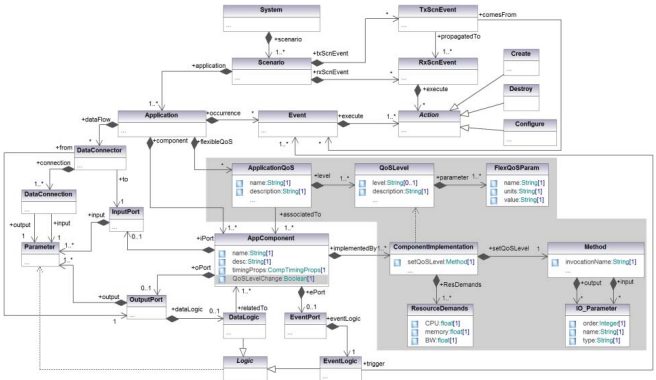
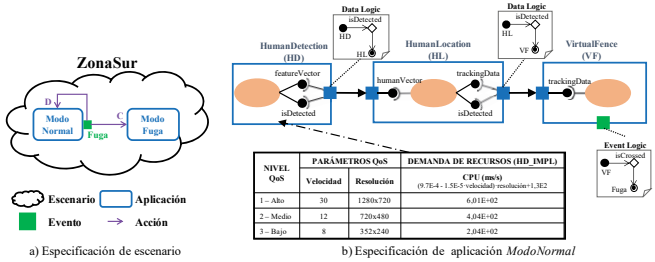


Fig. 1. Meta-Modelo para la especificación de aplicaciones

Por último, el concepto de Evento (*Event*) permite identificar cambios de contexto relevantes que demandan una reacción, como es el caso de la detección de un intruso. El componente encargado de detectar un cambio de contexto dispone de un puerto de eventos (*EventPort*) que contiene la lógica para el lanzamiento de dicho evento (*EventLogic*). La reacción frente a un cambio de contexto consiste en un conjunto de acciones (*Action*) ejecutadas tras su detección y que tienen como objetivo a la propia aplicación u otras aplicaciones. Es importante destacar que los eventos también pueden propagarse a aplicaciones de otros escenarios (*TxScnEvent* y *RxScnEvent*).

Siguiendo con el caso de estudio del control perimetral, se trata de un escenario constituido por dos aplicaciones, una por cada modo de operación. En la Fig. 2.a se puede observar que las aplicaciones están relacionadas por un evento llamado *Fuga*, lanzado cuando la aplicación *ModoNormal* detecta que un preso ha cruzado una línea virtual.

El evento lleva asociada la ejecución de dos acciones: iniciar la aplicación *ModoFuga* y detener la aplicación *ModoNormal*. La Fig. 2.b detalla el diseño de la aplicación *ModoNormal* que está formada por tres componentes. El componente *HumanDetection* (HD) cíclicamente captura el video de una cámara IP que apunta al muro del centro penitenciario, y lo analiza con el objetivo de detectar figuras humanas. Cuando esto ocurre, envía la información correspondiente a la figura detectada al componente *HumanLocation* (HL). Este componente comprueba si la figura se había detectado con anterioridad y si está en movimiento. En caso afirmativo, envía la información sobre su trayectoria al componente *VirtualFence* (VF) que analiza si se ha cruzado la línea virtual, en cuyo caso lanza el evento *Fuga*.



NIVEL QoS	PARÁMETROS QoS		DEMANDA DE RECURSOS (HD_IMPL)
	Velocidad	Resolución	CPU (ms/s) (9.7E-4 - 1.5E-5 velocidad) resolución+1,3E2
1 – Alto	30	1280x720	6,01E+02
2 – Medio	12	720x480	4,04E+02
3 – Bajo	8	352x240	2,04E+02

a) Especificación de escenario

b) Especificación de aplicación *ModoNormal*

Fig. 2. Especificación del caso de estudio *Control Perimetral* en zona sur

Con respecto a la demanda flexible de recursos, los nuevos elementos de modelado se han resaltado en la Fig. 1. En el caso de aplicaciones multimedia, la QoS relativa al campo de aplicación (*ApplicationQoS*) puede estar determinada por diferentes parámetros (*QoSParam*) tales como la velocidad del video (número de imágenes por segundo), su resolución (alto x ancho de la imagen en número de pixeles) o su modo de codificación (tasa de bits, constante o variable), entre otros [7], [11]. Cuando se trata de aplicaciones que pueden tolerar cierta degradación de la calidad, como es el caso de *ModoNormal*, el usuario puede identificar un rango de niveles de QoS aceptables ordenados por su grado de degradación (*QoSLevel*). De esta manera, cada nivel de QoS estará definido por valores concretos de cada parámetro. Una vez que la llamada QoS flexible has sido diseñada, es necesario identificar los componentes afectados (relación *associatedTo*).

Es habitual que el cambio de nivel de QoS lo realice un único componente. Así por ejemplo, al modificar la configuración de la cámara que genera el stream de video, se modificará el nivel de QoS de todos aquellos componentes que lo procesen. Por lo tanto, el desarrollador de software tiene que definir e implementar lo(s) método(s) cuya invocación provoquen el cambio de nivel (propiedades *QoSLevelChange* y *setQoSLevel* de los elementos *AppComponent* y *ComponentImplementation*, respectivamente). Por último, el desarrollador también debe especificar los recursos demanda-

dos por todas las implementaciones de los componentes afectados por la QoS flexible, para cada uno de los niveles (*ResourceDemand*). Dicha demanda de recursos se suele expresar en función del número de ciclos de CPU, de la carga máxima de memoria, y del ancho de banda estimado de las implementaciones de componente.

En el ejemplo ilustrado en la Fig. 2.b, hay una QoS de aplicación determinada por la velocidad del video y su resolución, asociada al componente *HD* y con tres niveles de flexibilidad: alto, medio y bajo. Suponiendo una única implementación para dicho componente (*HD_IMPL*), en la tabla se muestran los valores de los parámetros en cada nivel, así como los recursos demandados en términos de CPU. Por último, en este ejemplo el componente *HD* también es el encargado del cambio de nivel de QoS, lo que en última instancia consiste en cambiar el modo de operación de la cámara.

3 Soporte en Tiempo de Ejecución de la Demanda Flexible de Recursos

3.1 Requisitos del Middleware

El middleware es el encargado de asegurar la mejor calidad posible para las aplicaciones que se estén ejecutando. Por lo tanto, debe aprovechar la demanda flexible de recursos que presentan algunas de ellas para ajustar su demanda, es decir ajustar su nivel de QoS, a la disponibilidad de recursos en un instante concreto. Esto implica reducir o aumentar su nivel de calidad en caso de sobrecarga o subutilización, respectivamente.

Con este propósito, el middleware debe conocer la arquitectura del sistema, tanto la de las aplicaciones como la de la infraestructura. También debe conocer la disponibilidad y demanda de recursos en todo instante, para poder detectar situaciones de sobrecarga o subutilización. Por último, también debe proporcionar mecanismos para la elección del mejor nivel de calidad posible para todas las aplicaciones en ejecución, así como mecanismos para poder establecer dicho nivel.

3.2 Arquitectura del Middleware MAS-RECON

Con el propósito de cumplir con las demandas de adaptabilidad y disponibilidad de aplicaciones flexibles, en un trabajo previo de los autores se propuso una arquitectura de middleware basada en multi-agentes para la gestión de la ejecución de las aplicaciones, así como una plantilla de agentes para la implementación de los componentes de aplicación [10]. El presente trabajo extiende algunos módulos de dicho middleware dotándoles de nuevas funcionalidades que cubren la demanda flexible de recursos. La plantilla de agentes también se ha completado con nuevas utilidades de control.

En la Fig. 3 se observa que el middleware MAS-RECON se basa en JADE, un framework para el desarrollo y gestión de sistemas multi-agentes [12], que soporta aplicaciones distribuidas en entornos heterogéneos. La arquitectura de middleware pro-

puesta extiende JADE con nuevos módulos (ver pate superior de la Fig. 3), cada uno implementado por un agente. Por lo tanto, en tiempo de ejecución, habrá agentes de middleware correspondientes a módulos del middleware y agentes de aplicación correspondientes a implementaciones de componentes en ejecución.

Más concretamente, el *Middleware Manager* (MM) es el orquestador principal que gestiona información sobre el diseño y ejecución de todo el sistema, recogida en el llamado *System Repository*. La información relativa a la ejecución de las aplicaciones, a los eventos lanzados y a los nodos arrancados se gestiona de forma distribuida, mientras que la información acerca del diseño de las aplicaciones (relacionada con la aproximación de modelado del apartado anterior) y de los nodos se almacena de forma local.

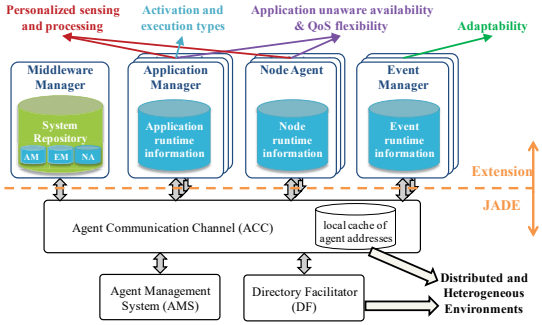


Fig. 3. Arquitectura de middleware basada en multi-agentes (MAS-RECON)

Hay una instancia del módulo *Node Agent* (NA) por cada nodo, que proporciona información sobre el diseño y el estado de ejecución del nodo. Esta información resulta útil para la gestión de recursos y para tareas de disponibilidad. Además, los NAs llevan a cabo el proceso de negociación para el despliegue de los agentes de aplicación en el nodo más adecuado. Existe también un *Application Manager* (AM) por cada aplicación arrancada, que gestiona la ejecución de sus componentes, lo que incluye: supervisar el proceso de negociación entre NAs para acoger instancias de dichos componentes, así como gestionar su estado de ejecución, necesario en procesos de recuperación con estado. Por último, cuando se detecta un cambio de contexto relevante, se inicia una instancia del módulo *Event Manager* (EM) por cada evento lanzado. El EM gestiona todas las acciones relacionadas con su evento.

En resumen, se puede concluir que el requisito de adaptabilidad es atendido por el módulo EM, mientas que la disponibilidad es soportada por los módulos AM y NA, tal y como se explica en [10]. Con respecto a la demanda flexible de recursos, el si-

guiente apartado describe la extensión de los módulos AM y NA, así como de la estructura del System Repository.

3.3 Gestión en Tiempo de Ejecución de Aplicaciones Multimedia Flexibles

Durante la ejecución de las aplicaciones pueden darse situaciones en las que la cantidad de recursos disponibles varíe. En ocasiones, la disponibilidad decrecerá debido al arranque de nuevas aplicaciones o a la caída de nodos. Por el contrario, el arranque de nuevos nodos o la detención de la ejecución de aplicaciones pueden dar lugar a un incremento en la disponibilidad de recursos. Por ejemplo, en el caso de estudio propuesto, suponiendo que cada instancia de los tres componentes de la aplicación *ModoNormal* (HD, HL y VF) se ejecuta en un nodo diferente (N_1, N_2 y N_3, respectivamente), la Fig. 4 muestra el proceso de detección de fallo de la instancia del componente HL (*HL_001*) y su correspondiente recuperación. En efecto, cuando la instancia de su componente previo (*HD_001*) detecta el fallo, avisa al AM correspondiente (*AM_MN*), quien supervisa el proceso de recuperación manteniendo el estado de la instancia fallida. Este proceso implica alojar una nueva instancia (*HL_002*) en otro nodo (N_1) mediante un proceso de negociación entre todos aquellos nodos que la puedan acoger (N_1 y N_3)

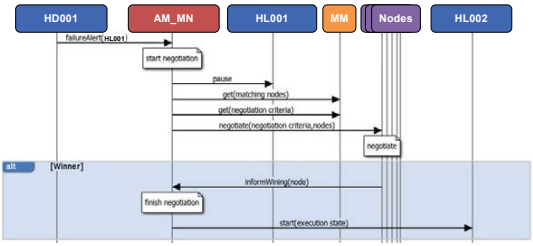


Fig. 4. Detección de fallo y recuperación con estado en la aplicación *ModoNormal*

Para una correcta gestión de la demanda flexible de recursos, se ha dotado a los NAs de mecanismos que continuamente monitorizan el consumo de recursos en su nodo (ciclos de CPU, carga de memoria y ancho de banda). Si detectan que dicho consumo se encuentra por encima o por debajo de un determinado umbral durante cierto periodo de tiempo, informan al MM. Continuando con el ejemplo anterior, en la Fig. 5.a se observa que el haber acogido a una nueva instancia puede dar lugar a una situación de sobrecarga de CPU en el nodo N_1, situación que es detectada por su NA (*NA_N1*). Este NA lanza un evento de aviso que llega hasta el MM a través de todas los AMs relacionados con instancias que estén ejecutándose en dicho nodo. Por su parte, el MM dispone de mecanismos que le permiten evitar atender el mismo evento procedente de AMs diferentes. Por simplicidad, en este ejemplo existe una única aplicación, y por lo tanto un único AM (*AM_MN*).

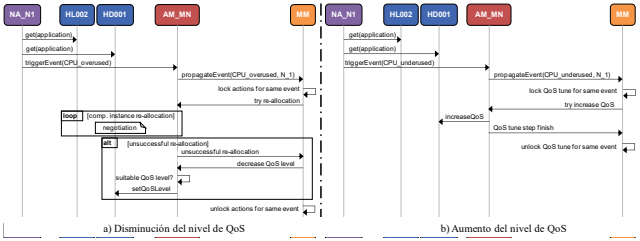


Fig. 5. Demanda flexible de recursos en aplicaciones multimedia

Cuando recibe un evento de este tipo, el MM intenta reducir la sobrecarga del nodo de dos maneras. En primer lugar, intenta realojar las instancias que estén corriendo en dicho nodo, una a una, hasta que la carga de CPU se encuentre por debajo del umbral establecido, siguiendo el proceso de negociación descrito en la Fig. 4. En caso de que algún realojamiento no tenga éxito, el MM comprueba si existen aplicaciones con demanda de recursos flexible, y en ese caso inicia el cambio a un nivel de QoS menor. Para ello, se ha extendido la estructura del System Repository con información del diseño de la QoS flexible de las aplicaciones (nuevos elementos en color naranja en la Fig. 6). Además, la ontología de comandos - descrita en un trabajo previo [10] para permitir que los módulos del middleware interactúen con los agentes de aplicación – se ha extendido con un nuevo comando para modificar el nivel de QoS de una instancia, el método setQoSLevel. En el ejemplo de la Fig. 5.a, en caso de que una negociación no tenga éxito, el *AM_MN* reduce el nivel de QoS de su aplicación, enviando el comando de control setQoSLevel a la instancia del componente HD (*HD_001*).

De forma similar, si los recursos del sistema están subutilizados, el middleware MASRECON gestiona el cambio al nivel de QoS más alto posible. Tal y como se describe en la Fig. 5.b, cuando el NA del nodo N_1 (*NA_N1*) detecta que su uso de CPU es inferior a un determinado umbral, lanza un evento de aviso que alcanza al MM a través de los AMs relacionados con instancias que estén corriendo en dicho nodo (*AM_MN* en Fig. 5.b). El MM comprueba si existen aplicaciones con demanda flexible de recursos y en ese caso inicia el proceso de cambio a su siguiente nivel superior. Este proceso se repite tantas veces sea necesario hasta que el uso de recursos del nodo se recupere. Resulta importante destacar, que este proceso de modificación del nivel de QoS puede llevar a un bucle infinito de aumento-disminución entre dos niveles consecutivos. Es por ello que el MM está provisto de mecanismos que evitan más de una iteración de aumento-disminución entre dos niveles consecutivos.

4 Resultados Experimentales

En esta sección se presenta la viabilidad de la propuesta para hacer frente a la demanda de recursos flexible de algunas aplicaciones multimedia. Con respecto al diseño, el

sistema consta de tres escenarios (zona sur, zona norte y zona este), todos ellos definidos según la especificación mostrada en la Fig. 2.a. Las aplicaciones necesarias (*ModoNormal_S*, *ModoFuga_S*, *ModoNormal_N*, *ModoFuga_N*, *ModoNormal_E* y *ModoFuga_E*) se han diseñado siguiendo la aproximación de modelado descrita en la Sección 2, y como muestra, la Fig. 2.b presenta el diseño de la aplicación *ModoNormal_S*. Este diseño tiene en cuenta la demanda flexible de recursos de las aplicaciones que se corresponde con flexibilidad en la QoS del nivel de aplicación. Siguiendo la metodología de desarrollo descrita en [10] se ha desarrollado una implementación, es decir un agente, para cada componente, y se ha registrado la información de diseño necesaria en el System Repository (ver Fig. 6). El rendimiento en tiempo de ejecución del middleware MAS-RECON se ha evaluado con respecto a su capacidad para asegurar el mejor nivel de QoS para las aplicaciones en ejecución. Para ello se ha realizado la prueba experimental presentada en la Fig. 7.

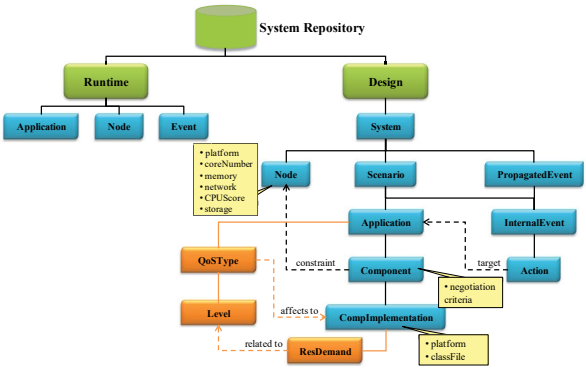


Fig. 6. Estructura extendida del System Repository

La infraestructura del sistema está formada por dos únicos nodos (*Nodo1* y *Nodo2*). Los tres escenarios se inician en su modo de operación normal (instante I_0), lo cual implica el arranque de tres aplicaciones flexibles: *ModoNormal_S*, *ModoNormal_N*, y *ModoNormal_E*. El *Nodo1* aloja todas las instancias de las aplicaciones *ModoNormal_S* y *ModoNormal_N*, mientras que el *Nodo2* las de la aplicación *ModoNormal_E* y la aplicación *ModoFuga_S*, que se activará por evento. Este despliegue de instancias ha sido fijado por medio de restricciones a nodo definidas en el System Repository, con el objetivo de simplificar el test, enfocándolo en la gestión de niveles de QoS.

Como las aplicaciones se arrancan en su nivel de QoS más bajo, en la Fig. 7 se puede observar que el *Nodo2* lanza dos eventos de subutilización (uso de CPU < 50%, *CPU_underused*), lo cual provoca un aumento en su demanda de CPU, en I_1 e I_2. Del mismo modo, en el *Nodo1* se observa que desde el instante I_0 hasta el I_3 el uso de CPU

va en aumento, lo que también se corresponde con incrementos de QoS de sus aplicaciones. Es justamente en I_3 cuando el *Nodo1* lanza un evento de sobrecarga de CPU (uso de CPU > 90%, *CPU_oversused*), que se gestiona como en la Fig. 5.a. En este caso, las restricciones a nodo imposibilitan el realojo de las instancias, por lo que se reduce el nivel de QoS de la aplicación que se encuentre en el nivel más alto (en la Fig. 7 se observa cómo decrece el uso de CPU).

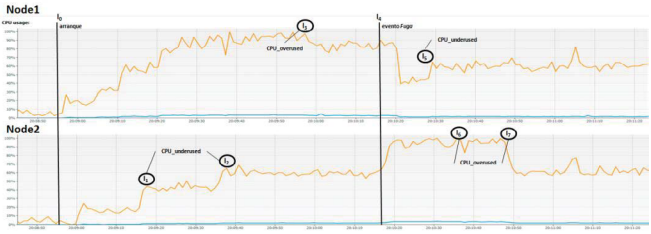


Fig. 7. Gestión de la flexibilidad de QoS del nivel de aplicación con respecto al uso de CPU

En el instante I_4, el componente VF de la aplicación *ModoNormal_S* detecta una posible fuga, lanzando el evento *Fuga*. Como consecuencia (ver Fig. 2.a) se arranca la aplicación *ModoFuga_S* (aumenta el consumo de CPU en *Nodo2*) y se detiene la aplicación *ModoNormal_S* (se reduce el consumo de CPU en *Nodo1*). Como resultado de estos cambios de demanda, el *Nodo1* lanza un evento de subutilización lo que provoca el aumento de la QoS de la aplicación *ModoNormal_N* (I_5). Por su parte, el *Nodo2* lanza dos eventos de sobrecarga que resultan en la bajada del nivel de la aplicación *ModoNormal_E* (I_6 e I_7). Nótese que el efecto del primero es inapreciable ya que a pesar de la reducción se continúa en situación de sobrecarga.

5 Conclusiones y Trabajo Futuro

Este artículo presenta una solución para el diseño y gestión de la demanda flexible de recursos de las aplicaciones multimedia. Se ha mostrado cómo la aproximación de modelado propuesta dispone de los mecanismos necesarios para que el experto de domino caracterice la flexibilidad de su QoS de aplicación, y para que el desarrollador de software defina la correspondiente demanda flexible de recursos. De hecho, esta es la información que el middleware MAS-RECON necesita para poder asegurar el mejor nivel de QoS de las aplicaciones en ejecución, teniendo en cuenta la disponibilidad de recursos en un instante concreto.

Mediante una prueba experimental con aplicaciones multimedia se ha demostrado que los mecanismos de los que disponen los módulos del middleware MAS-RECON son adecuados para poder gestionar su demanda flexible de recursos. Sin embargo, siempre han sido acciones reactivas, lo cual es inaceptable en caso de aplicaciones críticas

(la entrada de la aplicación *ModoFuga_S* provoca una situación de sobrecarga mantenida). Es por ello que actualmente se está trabajando en un algoritmo de control de admisión que asegure que únicamente se aceptan en el sistema aquellas aplicaciones cuyas demandas de recursos se puedan satisfacer, modificando previamente la calidad de las que están en ejecución, si fuera posible y necesario.

6 Agradecimientos

Este trabajo se ha subvencionado en parte por el Gobierno de España (MCYT) bajo el proyecto DPI- 2015-68602-R y por la Universidad del País Vasco (UPV/EHU) con la subvención UFI11/28.

7 Referencias

1. Sadri , F.: Ambient intelligence: A Survey. ACM Comput. Surv. 43, 4, 1–66 (2011)
2. Bajo, J., Fraile, J. A., Pérez-Lancho, B., Corchado, J. M.: The THOMAS architecture in Home Care scenarios: A case study. Expert Syst. Appl., 37, 5, 3986–3999 (2010)
3. Valls, M. G., López, I. R., Villar, L. F.: iLAND : An Enhanced Middleware for Real-Time Reconfiguration of Service Oriented Distributed Real-Time Systems. IEEE Trans. Ind. Informatics, 9, 1, 228–236 (2013)
4. Tsadimas, A.: Model-based enterprise information system architectural design with SysML. In: 9th IEEE International Conference on Research Challenges in Information Science (RCIS), pp. 492–497. IEEE, Athens (2015)
5. Estévez-Ayres, I., Basanta-Val, P., García-Valls, M., Fisteus, J. A., Almeida, L.: QoS-aware real-time composition algorithms for service-based applications. IEEE Trans. Ind. Informatics, 5, 3, 278–288 (2009)
6. de Souza Neto, P. A.: A Methodology for Building Service-Oriented Applications in the Presence of Non-Functional Properties. Universidade Federal do Rio Grande do Norte, 2012.
7. Bellavista, P., Corradi, A., Stefanelli, C.: Application-level QoS control for video-on-demand. IEEE Internet Comput., 7, 6, 16–24 (2003)
8. Brandt, S., Nutt, G., Berk, T., Mankovich, J.: A dynamic quality of service middleware agent for mediating application resource usage. In: IEEE Real-Time Syst. Symp., pp. 307–317. IEEE, Madrid (1998)
9. Cucinotta, T., Palopoli, L., Abeni, L., Faggioli, D., Lipari, G.: On the integration of application level and resource level QoS control for real-time applications. IEEE Trans. Ind. Informatics, 6, 4, 479–491 (2010)
10. Armentia, A., Gangoiti, U., Priego, R., Estévez, E., Marcos, M.: Flexibility support for homecare applications based on models and multi-agent technology. Sensors, 15, 12, 31939–31964 (2015)
11. Chalmers, D., Sloman, M.: A survey of quality of service in mobile computing environments. IEEE Commun. Surv. Tutorials, 2, 2, 2–10(1999)
12. Bellifemine, F., Caire, G., Poggi, A., Rimassa, G.: JADE: A software framework for developing multi-agent applications. Lessons learned. Inf. Softw. Technol., 50, 1–2, 10–21 (2008)

Modelado, análisis temporal, configuración y optimización

Obtención del WCET óptimo con caches de instrucciones bloqueables (Lock-MS) en Otawa*

Alba Pedro-Zapater†, Clemente Rodríguez‡, Juan Segarra†, Rubén Gran†, y Víctor Viñals-Yúfera†

†Dpt. Informática e Ingeniería de Sistemas, Universidad de Zaragoza, España
Instituto de Investigación en Ingeniería de Aragón (I3A), U. Zaragoza, España
‡Dpt. Arquitectura y Tecnología de Computadores, U. País Vasco, España
†‡Red de Excelencia HiPEAC-3 (European FET FP7/ICT 287759)
†{albapz,jsegarra,rgran,victor}@unizar.es, ‡acprolac@ehu.es

Resumen El WCET de un programa es difícil de calcular debido a la falta de predictibilidad de las caches convencionales. En este trabajo hemos implementado un módulo de análisis del WCET desde el binario del programa para la herramienta Otawa para una cache bloqueable con el método *Lock-MS*. Este módulo automatiza la creación de las restricciones ILP para el cálculo del WCET y de las lineas seleccionadas de la cache bloqueable. Esta automatización nos permite disponer de un entorno de experimentación productivo y de amplia aplicabilidad. En este trabajo hemos utilizado este entorno para estudiar el impacto en el WCET de los niveles de optimización, la configuración de cache y la latencia a memoria. Los resultados obtenidos muestran las posibilidades del uso de este módulo para benchmarks más grandes y complejos.

Keywords: WCET, Cache de Instrucciones, Lock-MS, Otawa

1. Introducción

Uno de los principales retos en el estudio de los sistemas de tiempo real estricto (HRTS: *Hard Real Time Systems*) es el cálculo del tiempo de ejecución en el peor caso (WCET: *Worst Case Execution Time*) o en su defecto una cota superior lo más ajustada posible. Aunque existen métodos probabilísticos que calculan el WCET basándose en medidas realizadas sobre ejecuciones concretas, dichos métodos no garantizan formalmente que los valores obtenidos sean una cota superior del WCET (e.g. [5,6]). Para garantizar formalmente dichas cotas, es necesario utilizar métodos de análisis estático (e.g. [3,7,9,11]). Estos métodos se basan en el análisis del código del programa y en su grafo de flujo de control, partiendo del código fuente o desde el binario. El análisis del código fuente es

* Este trabajo ha sido parcialmente financiado por los proyectos TIN2013-46957-C2-1-P, Consolider NoE TIN2014-52608-REDC (Gob. España), gaZ: grupo de investigación T48 (Gob. Aragón y European ESF), Unizar (JIUZ-2015-TEC-06), y la beca FPU14/02463.

problemático ya que el compilador puede transformar el flujo original del programa (optimizaciones). En cambio, el análisis del binario puede suponer perder información semántica que está presente en el código fuente.

Una de las principales dificultades para obtener un WCET preciso mediante el análisis del código es que algunos componentes hardware tienen una latencia variable, por ejemplo la jerarquía de memoria. Aunque las memorias cache convencionales reducen significativamente el tiempo medio de acceso a memoria, el coste de cada acceso es difícil de predecir ya que depende de los accesos anteriores y los parámetros de cache (reemplazo, asociatividad, tamaño, etc.). Como resultado, la diferencia entre el WCET real y la cota calculada puede ser excesiva.

Para evitar la dificultad de predecir de forma precisa el comportamiento de las caches convencionales, una propuesta es usar caches cuyo contenido está prefijado (lock-caches) y su mecanismo de reemplazo está deshabilitado. De esta forma, dado que el contenido es conocido y no cambia, el cálculo de aciertos y fallos es trivial y no depende de la secuencia de accesos. El inconveniente de fijar los contenidos en cache es, por una parte determinar buenos contenidos a fijar, y por otra parte que al no actualizarse los contenidos en ejecución se va a limitar la explotación del reuso temporal y espacial de la aplicación. En el caso de la cache de instrucciones, se ha demostrado que el uso de un *line-buffer* es analizable para el cálculo de WCET y permite adaptar el contenido en la jerarquía de memoria al flujo de ejecución del programa [3,13]. El *line-buffer* tiene el tamaño de un bloque de cache y actúa como una cache convencional de un solo bloque.

Lock-MS es un método análisis estático de WCET sobre caches bloqueables [3]. Este método, además de calcular el WCET, obtiene el contenido óptimo de la cache bloqueable. Es decir, dada cierta cache bloqueable, obtiene el contenido que minimiza el WCET para cierta tarea. Este método se basa en generar un modelo de programación lineal entera (ILP) donde los grados de libertad del modelo son, para cada uno de los bloques de memoria, precargarlo en cache o no. Además de para caches de instrucciones bloqueables, este método ha sido extendido para analizar otros componentes hardware tales como prebuscadores de instrucciones [2], caches de datos específicas para tiempo real [14,15] y también para optimizar el WCET teniendo en cuenta el consumo energético en el peor caso [8].

Una de las mayores desventajas de los métodos de análisis estático es la falta de herramientas automáticas que los implementen. Esto se debe a que los métodos a implementar son relativamente recientes y orientados a campos muy especializados, y las implementaciones no suelen reconocerse como méritos científicos. Además, los métodos de análisis estático suelen ser métodos formales complejos y requieren información difícil de conseguir a partir de un simple fichero ejecutable.

En este trabajo, hemos implementado un módulo de análisis del WCET desde el binario del programa para la herramienta Otawa [4]. Este módulo es capaz de generar las restricciones ILP (*Integer Linear Programming*) del método Lock-MS [3], que permiten calcular el contenido de la cache de instrucciones que

minimiza el WCET del programa. A diferencia de herramientas anteriores, conociendo las cotas en el número de iteraciones de los bucles, el análisis y generación de las restricciones ILP se realiza de manera automática. Esto permite analizar el WCET de un gran número de experimentos de forma productiva y sencilla variando tanto métricas hardware como software. A resaltar que vamos a poder observar la influencia del compilador en el WCET, algo poco usual debido a la dificultad de trabajar con optimizaciones.

La estructura del articulo es la siguiente: en la Sección 2, hacemos una breve introducción a la herramienta de Otawa y se explica el módulo implementado para la generación de las restricciones ILP; en la Sección 3 presentamos nuestro entorno experimental; en la Sección 4 presentamos los resultados de los experimentos realizados y finalmente en la Sección 5 recogemos las conclusiones y presentamos el trabajo futuro.

2. Otawa

Otawa es un entorno de análisis estático de binarios desarrollado por el grupo TRACES del *Institut de Recherche en Informatique* de Toulouse [4]. Otawa permite analizar programas ejecutables de múltiples arquitecturas (PowerPC, ARM, SPARC o M68HCS) con el objetivo de analizar su WCET en presencia de diferentes estructuras de procesador y de memoria cache. Para ello Otawa genera el grafo del flujo de control (CFG, *Control Flow Graph*) que incorpora la información de todos los bloques básicos de instrucciones y sus relaciones de precedencia. En la Figura 1 podemos ver un ejemplo del CFG que genera Otawa para un pequeño código de ejemplo.

Todas las técnicas de análisis del WCET incorporadas en Otawa se basan en enriquecer y manipular el CFG para alcanzar su objetivo. Cada módulo de Otawa es un paso de transformación que va añadiendo información al CFG y resultados parciales. Esta estructura modular permite combinar módulos existentes (detección de bucles en el CFG, por ejemplo) con módulos nuevos. Estos módulos también aportarán lo necesario para la formulación de las restricciones ILP con las que se calcula el WCET del método *Lock-MS*.

2.1. Implementación del módulo Lock-MS

El módulo que hemos implementado en Otawa analiza el CFG de un binario y elabora el sistema ILP correspondiente. Su resolución proporciona tanto el WCET del programa como los bloques de cache que deben seleccionarse para la cache bloqueable de Instrucciones. El sistema ILP se genera siguiendo el método *Lock-MS*, lo cual asegura que el WCET obtenido es el menor posible con el mínimo número de bloques en la cache bloqueable [3]. En la Figura 2 podemos ver el grafo enriquecido por nuestro módulo del programa de la Figura 1 (a). El módulo *Lock-MS* realiza una búsqueda en profundidad en el CFG para obtener cada uno de los caminos posibles (dos caminos en el ejemplo de la Figura 2 (a)). Después calcula el número máximo de veces que se va a ejecutar cada bloque

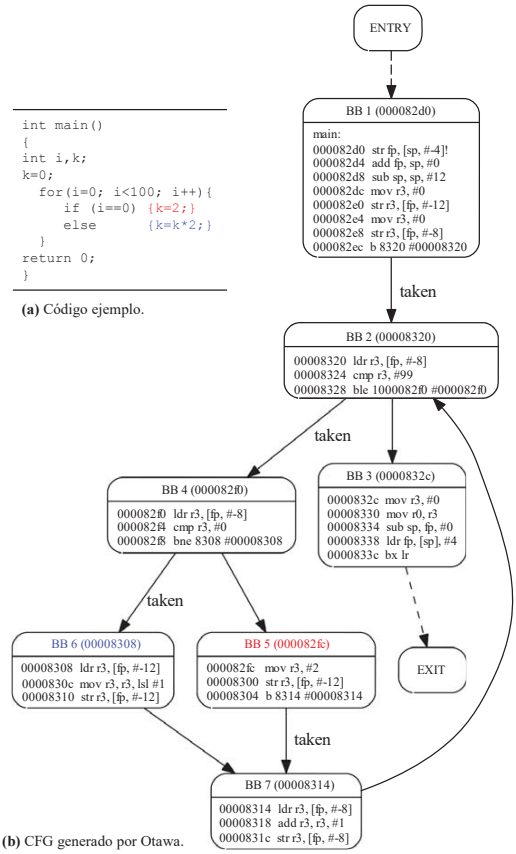


Figura 1. Ejemplo de código fuente (a) junto con su representación CFG de OTAWA a partir de un binario ARM (b).

básico que compone el CFG (por ejemplo el bloque básico 6 (BB6) se ejecuta un máximo de 100 veces). También se calcula toda la información referente a la configuración de la cache (tamaño de bloque, tamaño de cache y asociatividad), como podemos ver en la Figura 2 (a) donde aparece la información sobre a qué bloque de memoria pertenece cada instrucción y a qué conjunto pertenece cada bloque. Por ejemplo, en el bloque básico 6 (BB 6), la última instrucción pertenece al bloque 4 al cual le corresponde el conjunto 0 (S0 L4).

Con toda esta información se construye el sistema ILP y la representación gráfica del CFG que estamos estudiando, que nos facilitará el análisis de los datos obtenidos. En la Figura 2 (b) se muestra parte del sistema ILP correspondiente al programa de ejemplo. Podemos ver las restricciones correspondientes a cada camino y aquellas correspondientes a la configuración de cache; el conjunto al que pertenece cada bloque, y cuantos bloques puede haber en cada conjunto (uno en este caso).

Finalmente, como ya señalábamos, la resolución de este sistema nos proporciona tanto el WCET final, como los bloques que deben ser cacheados para conseguir un configuración óptima (WCET mínimo). Cabe destacar que este sistema minimiza también el número de bloques seleccionados.

2.2. Reducción del número de caminos

Algunos de los benchmarks analizados en este trabajo por su tamaño o por su complejidad, presentan un número muy alto de caminos (*crc*, *integral*, y *qurt*, ver Sección 3). En estos casos hemos contabilizado cientos de caminos, llegando a más de 50000 con nivel de optimización -O0 en *qurt*. Como consecuencia el tiempo necesario para resolver el sistema ILP aumenta considerablemente.

Para estos casos, hemos hecho una modificación del módulo para reducir el número de caminos, y por lo tanto el número de restricciones y complejidad del sistema ILP. El número de caminos aumenta de forma exponencial debido a la existencia de condicionales concatenados. En ciertos casos, el CFG original puede verse como varios subgrafos concatenados (cada uno de ellos con un único nodo de entrada y otro de salida). En esos casos, las propiedades de las caches bloqueadas hacen que el WCET global sea igual a la suma de los WCETs de cada subgrafo [3]. Siendo así, la modificación consiste en compactar el sistema ILP analizando los subgrafos independientemente para luego sumar los WCETs resultantes de cada uno, con lo que se reduce el tiempo para su resolución. La mejora será más o menos significativa dependiendo del número de caminos y complejidad del programa, y de los subgrafos que se hayan podido detectar. La división en subgrafos ha reducido notablemente el coste tal y como se puede ver en la Tabla 1 para *crc* compilado con -O0. Cabe destacar que incluso *qurt* compilado con -O0 no se puede analizar sin compactar caminos debido a que, como ya hemos dicho, tiene más de 50000 caminos.

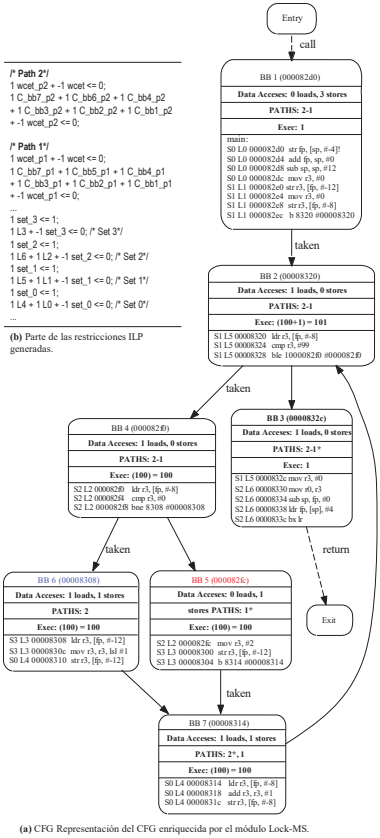


Figura 2. (a) CFG enriquecido del programa de la Figura 1 con los caminos numerados, el número máximo de ejecuciones por bloque básico, el bloque y conjunto de cache de cada instrucción y el número de instrucciones de datos. (b) Parte de las instrucciones ILP generadas por el módulo para este CFG. (a) y (b) tienen la siguiente configuración de cache: Tamaño Bloque 16B, Tamaño Cache 64B y Asociatividad 1.

Tiempo (s)	Calculando Caminos	Construyendo ILP	Resolviendo ILP
No compacto	0,036	3,925	162,62
Compacto	0,002	0,009	0,17

Tabla 1. Tiempos de procesamiento del WCET para *crc* con nivel de optimización -O0 medido en un portátil con un procesador Intel CORE i7

3. Entorno experimental

Para nuestros experimentos hemos utilizado parte de la colección SNU-RT Benchmark Suite for Worst Case Timing Analysis [16]: *jfdctint*, *lms*, *crc*, *matmult*, *integral*, *qurt* y *fibcall*. Además hemos utilizado dos benchmarks propios: *Matmult_opti* que es una versión optimizada en cuanto al acceso a datos de *matmult* y *gauss* que resuelve un sistema de ecuaciones lineales por el método de Gauss. Todos los límites de los bucles de estos benchmarks son conocidos, y no hay recursión.

Los binarios están generados para la arquitectura ARM v5t, usando el compilador cruzado *arm-none-eabi-gcc* versión 4.8.4 con distintos niveles de optimización. Este compilador cruzado considera por defecto que las operaciones con números reales se hacen por software. Sin embargo nosotros forzamos la generación de aritmética con instrucciones de coma flotante. También se ha configurado el compilador para evitar las sustituciones de parte del código con funciones de bibliotecas optimizadas (e.g. que sustituya asignaciones a cero en bucles por *memset*). Esto nos permite acotar adecuadamente los bucles al impedir el uso de funciones externas desconocidas. Respecto al procesador asumimos una segmentación ideal de dos etapas (E1: búsqueda de instrucción; E2: ejecución y lectura/escritura de operandos/resultados), añadiendo un ciclo a las instrucciones de memoria para que tengan más peso. También asumimos que no hay otros componentes con latencia variable (cache de datos, predictor de saltos, ejecución fuera de orden. etc.) más allá de la cache de instrucciones. Muchos sistemas para tiempo real pueden modelarse así, o bien porque no disponen de los mecanismos citados, o bien porque estos mecanismos se desconectan para impedir su variabilidad temporal. Por ejemplo el procesador Leon 4 permite desconectar su predictor de saltos [12]. Estas consideraciones también se asumen en estudios previos [10,13].

Para demostrar la capacidad de análisis del nuevo módulo lock-MS hemos realizado un barrido experimental amplio de opciones software y hardware. Las variables de experimentación y sus rangos son los siguientes:

- Niveles de optimización en compilación: -O0, -O1, -O2 y -O3.
- Sistema cache bloqueada y *line-buffer*:
  - Tamaño de bloque: 16, 32 y 64 Bytes.
  - Tamaño de cache bloqueada: Sin cache (solo *line-buffer*), infinita, 16, 32, 64, 128, 256, 512 y 1024 Bytes.

  - Asociatividad: 1, 2, 4 y completamente asociativa
- Acceso a datos:
  - 1 ciclo (e.g. datos precargados en cache/scratchpad)
  - Tiempo de acceso a memoria (sistema sin cache de datos)
- Acceso a memoria (fallo en instrucciones o acceso a datos): 1 (igual al coste de acierto), 5, 10 y 20 ciclos.

4. Resultados

Para cada configuración de la cache bloqueable, latencia de memoria y nivel de optimización hemos utilizado el módulo *Lock-MS* para calcular la selección de bloques a fijar con la que se obtiene el WCET óptimo. Ahora bien, puesto que los benchmarks analizados, por su pequeño tamaño, no son del todo representativos de tareas reales en aplicaciones actuales, no es intención de este trabajo extraer conclusiones sobre la interacción entre compilación y jerarquía de memoria. Este interesante objetivo se abordará en un trabajo posterior que considerará aplicaciones de prueba de mayor entidad como las contenidas en la colección TACLe [1].

Sin embargo el análisis de los resultados de estos benchmarks, aunque preliminar, aporta conclusiones significativas. Un caso interesante es *crc*. Las Tablas 2 y 3 muestran el comportamiento de *crc* variando la mayoría de los parámetros descritos. En estas tablas el tamaño de bloque es 16B, la latencia de accesos a datos es 1 ciclo y la latencia de fallo de instrucciones es 5 ciclos (excepto para el WCET ideal, que se especifica que es 1 ciclo).

En la Tabla 2 para cada nivel de optimización tenemos el número de bloques y de caminos en la segunda y tercera columna. A continuación, en la cuarta y quinta columna, tenemos el WCET ideal (latencia de acceso de instrucciones en caso de fallo 1 ciclo, ningún bloque de cache resulta seleccionado) y el WCET con cache bloqueable infinita. La diferencia entre ellos es que con la cache bloqueable infinita se tiene en cuenta el coste de cargar los bloques. En la última columna, aparece el número de bloques seleccionados para la cache infinita; ya que como nuestro sistema ILP minimiza el número de bloques cargados, este número no corresponde necesariamente al total de bloques del programa (columna 2). De esta tabla ya podemos extraer interesantes conclusiones sobre el sistema, ya que si el WCET requerido es menor que los mostrados en la tabla (en este caso el menor es el obtenido con nivel de optimización -O3) será necesario replantear el sistema. También observamos que el número de caminos y de bloques varía para cada nivel de optimización, y que un número mayor de bloques y caminos no implica un mayor WCET. Por ejemplo, tanto para el WCET ideal como para el WCET con cache infinita, el nivel de optimización -O3 tiene menor WCET que los niveles de optimización -O1 y -O2 pero tiene más caminos y más bloques.

En la tabla 3 cada columna corresponde a un tamaño de cache desde 0 hasta 64 bloques, y dentro de cada columna tenemos el WCET para cada nivel de optimización (O0-O3) agrupadas por las distintas asociatividades (1, 2, 4 y completamente asociativa). Para los tamaños de cache 32 y 64 (512 y 1024 Bytes) la

				Cache Bloqueable Infinita	
Opt	**Tamaño (bloques)**	**Núm. caminos**	**WCET ideal (Lat. mem.=1)**	**WCET**	**Núm. Bloques seleccionados**
O0	68	1296	203829	204144	55
O1	29	576	60361	60503	26
O2	23	576	54521	54637	20
O3	38	784	45943	46122	31

Tabla 2. En las columnnas: Tamaño (en bloques de 16 B), número de caminos posibles, WCET con latencia a memoria 1 (en ciclos), y WCET (en ciclos) y número de bloques seleccionados para la cache bloqueable de tamaño infinito y en las filas: distintos niveles de optimización del benchmark *crc*.

segunda columna recoge el número bloques seleccionados en la cache bloqueable; como ya pasaba en el caso de cache infinita, se seleccionan menos bloques que el total de bloques del programa (columna 2 de la tabla 2). Cuando no aparece esta segunda columna significa que se usan todos los bloques de la cache bloqueable.

Como ya hemos señalado nuestro sistema ILP minimiza el número de bloques en la cache, es decir, si tiene el mismo coste poner un bloque o no ponerlo, no lo pone. Esto nos ofrece la posibilidad de desconectar bloques de cache para ahorrar energía estática.

El WCET con cache infinita (columna 5 de la tabla 2) es una cota inferior de lo que se puede llegar a conseguir con una cache. El número de bloques seleccionados con cache infinita (columna 6 de la tabla 2) es el mínimo número de líneas necesarias en la cache para conseguir el mejor WCET, siempre y cuando no haya problemas de contención por usar una configuración de cache (número de conjuntos y grado de asociatividad) inadecuada para el benchmark. Estas cotas nos permiten verificar que los datos obtenidos son coherentes ya que se verifica que dado un nivel de optimización ningún WCET para ninguna configuración de cache de la tabla 3 es menor que el de la cache infinita (en negrita los que han alcanzado esa cota). De manera similar, ninguna selecciona un mayor número de bloques que los seleccionados en la cache infinita.

También podemos observar la influencia de la capacidad y de la asociatividad en el aumento del WCET. Por ejemplo para tamaño 16 (256B) con nivel de optimización -O1 observamos que hay aumento de WCET por debajo de asociatividad 4 (en cursiva los datos del ejemplo). De manera similar podemos ver la influencia de la capacidad. Por ejemplo, en el nivel de optimización -O2 y tamaños de cache menores que 32 (512B) nunca se alcanza el WCET obtenido con la cache infinita.

También hay que resaltar que el nivel de optimización -O3 no siempre es el mejor. Por ejemplo, en el caso del tamaño de cache 4 (64B) y asociatividad 1 el nivel de optimización -O2 es el que nos da un menor WCET. Esto implica que se deben estudiar todos los niveles de optimización para saber dónde se encuentra el menor WCET posible, aumentando el espacio de experimentación

		Tamaño de cache bloqueable, en número de bloques (de 16B)									
		0	1	2	4	8	16	32		64	
1-asoc	O0	376673	358246	339819	307061	269049	246849	224347	30	204147	54
	O1	144205	125770	107343	85697	78389	*70181*	**60503**	26	**60503**	26
	O2	128977	110550	92123	70477	63153	54649	**54637**	20	**54637**	20
	O3	91855	88636	86193	82083	73879	57471	46128	29	46125	30
2-asoc	O0			339819	307061	267873	246825	216461	32	204147	54
	O1			107343	85697	73125	*67837*	**60503**	26	**60503**	26
	O2			92123	70477	58777	54649	**54637**	20	**54637**	20
	O3			86193	82083	73879	57471	46128	29	**46122**	31
4-asoc	O0				307061	267865	244497	210341	32	204147	54
	O1				84825	73125	*60533*	**60503**	26	**60503**	26
	O2				70477	58777	54649	**54637**	20	**54637**	20
	O3				82083	73879	57471	46128	29	**46122**	31
Cpl-asoc	O0					267865	242177	209461	32	**204144**	55
	O1					71949	*60533*	**60503**	26	**60503**	26
	O2					58769	54649	**54637**	20	**54637**	20
	O3					73879	57471	**46122**	31	**46122**	31

Tabla 3. En las columnas, tamaño de la cache bloqueable en bloques, y en las filas el WCET (en ciclos) para cada nivel de optimización agrupadas por las distintas asociatividades. Si el número de bloques seleccionados es menor que los del programa, aparecen en una columna junto al WCET correspondiente

a analizar. Por último, generamos tablas similares a la anterior (no mostradas en este artículo) para valores distintos de latencia a memoria, tanto de instrucciones como de datos, y diferentes tamaños de bloque tanto para *crc* como para el resto de los benchmarks sin observar comportamientos cualitativamente diferentes a los descritos. Queda por analizar si con benchmarks más grandes y complejos se encontrarán comportamientos distintos.

5. Conclusiones y trabajo futuro

En este trabajo se ha presentado la implementación de un módulo de Otawa que permite analizar el WCET de binarios en presencia de una cache bloqueable junto a un *Line-Buffer*. Como resultado de este análisis se obtiene tanto el WCET como los bloques que se cargarán inicialmente en la cache bloqueable. Automatizar el proceso de análisis del WCET y de selección de los bloques nos permite barrer una gran cantidad de parámetros software y hardware para ver la sensibilidad del WCET. Debido al pequeño tamaño de los benchmarks analizados se ha realizado un análisis de poca profundidad que sin embargo nos permite hacernos una buena idea general de todo el potencial que puede tener el uso de este módulo para el posterior análisis de benchmarks con más entidad y complejidad.

Los resultados obtenidos resultan coherentes lo cual nos permite presumir la corrección del módulo. Además como aportación novedosa de este trabajo, hemos realizado un análisis de la influencia en el WCET del nivel de optimización aplicado por el compilador.

Como continuación de este trabajo, vamos a abordar los siguientes problemas: metodología a seguir en caso de benchmarks grandes e incluir en nuestro módulo el análisis de la jerarquía de memoria para datos.

5.1. Algoritmos para programas más grandes

Como ya hemos explicado en la Sección 2, uno de los problemas con los que nos hemos encontrado ha sido que debido al tamaño y/o complejidad, algunos benchmarks tienen un gran número de caminos. Esto tiene un impacto directo en el tiempo necesario para su enumeración y para la posterior resolución del sistema ILP asociado a todos estos caminos. Por ello, y para poder resolver benchmarks más grandes y complejos, estamos trabajando en un algoritmo basado en la teoría de grafos para simplificar el CFG detectando subgrafos. La versión del módulo utilizada en este trabajo permite incluir manualmente esta propuesta de división para los programas más complejos (Sección 2). En la versión final, vamos a ser capaces de automatizar la división del grafo y el posterior análisis de cada uno de los subgrafos. Para cada uno de estos subgrafos el número de caminos a considerar sería sustancialmente menor, facilitando la resolución del ILP. Esto también nos permitirá detectar distintas fases en el programa, pudiendo elegir distintos contenidos de la cache bloqueable para cada fase. Durante la ejecución del programa, estos puntos de cambios de fase serían candidatos para realizar la actualización del contenido de la cache bloqueable, y así, ajustar mejor su contenido a los requerimientos de la nueva fase y por lo tanto mejorar el WCET.

5.2. Integración de ACDC

De manera similar a cómo hemos planteado la integración de la cache bloqueable y el *line-buffer* para instrucciones en Otawa, se añadiría la cache de datos ACDC [15] para conseguir un análisis de WCET para accesos a memoria más preciso y completo. Además esto aportará una herramienta para que esta propuesta de diseño de cache de datos para tiempo real pueda ser evaluada con distintos benchmarks, caches de instrucciones, pipelines, etc.

Referencias

1. Timing analysis on code-level (tacle). http://www.tacle.eu/.
2. L. C. Aparicio, J. Segarra, C. Rodríguez, and V. Viñals. Combining prefetch with instruction cache locking in multitasking real-time systems. In *Proceedings of the IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications*, pages 319–328, Macau SAR, China, August 2010. IEEE Computer Society Press.

3. L. C. Aparicio, J. Segarra, C. Rodríguez, and V. Viñals. Improving the WCET computation in the presence of a lockable instruction cache in multitasking real-time systems. *Journal of Systems Architecture*, 57(7):695–706, August 2011.
4. Clément Ballabriga, Hugues Cassé, Christine Rochange, and Pascal Sainrat. Otawa: An open toolbox for adaptive wcet analysis. In *Proceedings of the 8th IFIP WG 10.2 International Conference on Software Technologies for Embedded and Ubiquitous Systems*, SEUS'10, pages 35–46, Berlin, Heidelberg, 2010. Springer-Verlag.
5. Guillem Bernat, Antoine Colin, and Stefan M. Petters. WCET analysis of probabilistic hard real-time systems. In *Proceedings of the 23rd Real-Time Systems Symposium RTSS 2002*, pages 279–288, 2002.
6. Francisco J. Cazorla, Eduardo Quiñones, Tullio Vardanega, Liliana Cucu, Benoit Triquet, Guillem Bernat, Emery Berger, Jaume Abella, Franck Wartel, Michael Houston, Luca Santinelli, Leonidas Kosmidis, Code Lo, and Dorin Maxim. Proartis: Probabilistically analyzable real-time systems. *ACM Trans. Embed. Comput. Syst.*, 12(2s):94:1–94:26, May 2013.
7. S. Ghosh, M. Martonosi, and S. Malik. Cache miss equations: A compiler framework for analyzing and tuning memory behavior. *ACM Transactions on Programming Languages and Systems*, 21(4):703–746, 1999.
8. R. Gran, J. Segarra, C. Rodríguez, L. C. Aparicio, and V. Viñals. Optimizing a combined wcet-wcec problem in instruction fetching for real-time systems. *Journal of Systems Architecture*, 59(9):667–678, October 2013.
9. Y. T. S. Li, S. Malik, and A. Wolfe. Cache modeling for real-time software: beyond direct mapped instruction caches. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 254–264, December 1996.
10. A. Martí Campoy, Á. Perles Ivars, and J. V. Busquets Mataix. Static use of locking caches in multitask preemptive real-time systems. In *IEEE Real-Time Embedded System Workshop*, December 2001.
11. F. Mueller. Timing analysis for instruction caches. *Real-Time Systems*, 18(2-3):217–247, May 2000.
12. Quad Core LEON4 SPARC V8 Processor. Data Sheet and User's Manual. `http://www.gaisler.com/doc/LEON4-N2X-DS.pdf`, 2015. [Online; accessed 21-Juny-2016].
13. I. Puaut and D. Decotigny. Low-complexity algorithms for static cache locking in multitasking hard real-time systems. In *Proc. of the IEEE Real-Time Systems Symp.*, December 2002.
14. J. Segarra, C. Rodríguez, R. Gran, L. C. Aparicio, and V. Viñals. A small and effective data cache for real-time multitasking systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 45–54, Beijing, China, April 2012.
15. Juan Segarra, Clemente Rodríguez, Rubén Gran, Luis C. Aparicio, and Víctor Viñals. ACDC: Small, predictable and high-performance data cache. *ACM Trans. Embed. Comput. Syst.*, 14(2):38:1–38:26, February 2015.
16. Seoul National University Real-Time Research Group. SNU-RT benchmark suite for worst case timing analysis, 2008.

Herramienta de configuración para sistemas IMA-SP

YOLANDA VALIENTE, PATRICIA BALBASTRE, JOSÉ ENRIQUE SIMÓ [1]

INSTITUTO DE AUTOMÁTICA E INFORMÁTICA INDUSTRIAL (AI2)

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Resumen

En este trabajo se describe la definición, diseño, implementación y prueba del conjunto de herramientas necesarias para realizar las actividades del proceso de desarrollo IMA-SP (Integrated Modular Avionics for SPace). Además, se definirá el modelo de datos, con los archivos asociados, que describen la configuración completa de un sistema con particiones y la evaluación de la viabilidad del sistema; el desarrollo de un prototipo del conjunto de herramientas, que se llamará IMA-SP SDT (System Development Toolkit) y la demostración de la herramienta sobre un caso de estudio.

1.-Introducción

Para hacer frente a la creciente complejidad del software de un satélite, la ESA ha explorado soluciones tecnológicas adoptadas por el dominio aeronáutico para este propósito: la arquitectura IMA Integrated Modular Avionics y los sistemas particionados (TSP).

En [1] se definen las actividades y los roles que intervienen en el proceso de desarrollo de un sistema IMA-SP. Dentro de estas actividades, tenemos: i) Particionado y asignación de recursos y ii) Evaluación de la viabilidad del Sistema.

La actividad de Particionado y asignación de recursos es una de las actividades centrales desde la perspectiva de definir las particiones de un sistema IMA-SP . En esta actividad, se realiza la asignación de componentes de software a nodos físicos teniendo en cuenta: las limitaciones de memoria y tiempo; requisitos de acceso directo a hardware específico; requisitos que afectan a las características del sistema, tales como el intercambio de información entre los componentes.

A lo largo del desarrollo del sistema ha de asegurarse que la asignación de recursos cumple con los requisitos establecidos y las limitaciones del hardware. Esto se puede comprobar mediante, por ejemplo, el análisis estático o mediante simulaciones. Esto incluye comprobar que los requisitos no funcionales del sistema, la planificación tem-

[1] Este articulo ha sido subvencionado por el Ministerio de Economía y Competitividad bajo los proyectos TIN2014- 56158-C4-1-P y TIN2014- 56158-C4-4-P.

poral, el control de fallos o la comunicación se cumplen tanto por la partición como por el hardware seleccionado. La evaluación de la viabilidad del sistema se lleva a cabo por el Integrador del Sistema (SI).

2.-Objetivos

El objetivo general de este artículo es definir , diseñar, implementar y probar el conjunto de herramientas necesarias para desarrollar un sistema IMA-SP. Para ello se definen los siguientes sub-objetivos:

1. Definir un modelo de datos que describa la información que necesita ser capturada para la definición de una partición, asignación de recursos, configuración de la plataforma, etc, es decir, reflejar la configuración completa del sistema con particiones, así como permitir la evaluación de la viabilidad del sistema.
2. Definir un conjunto apropiado de archivos de configuración y formatos de archivo correspondientes para capturar el contenido del modelo de datos, teniendo en cuenta el intercambio de información junto con la confidencialidad según se define en el proceso de desarrollo IMA-SP [1].
3. Desarrollar un prototipo del conjunto de herramientas llamado IMA-SP System Design Toolkit (SDT).
4. Demostrar la viabilidad del prototipo desarrollado con un caso de estudio.

3.-Modelo de datos

El modelo de datos a definir debe incluir la información necesaria para definir los recursos de hardware, los requisitos de la aplicación, la asignación de recursos y la configuración del kernel de particionado (PK). La información que es específica de los contenidos internos de las particiones (por ejemplo, la funcionalidad y la semántica) no está incluida. Los requisitos del modelo de datos son:

- El modelo de datos debe capturar los requisitos de las aplicaciones, como por ejemplo: requisitos temporales, necesidades de memoria o requisitos específicos de hardware.
- El modelo de datos debe capturar las propiedades de la plataforma: procesamiento, memoria, comunicación, temporizadores, recursos compartidos, etc.
- El modelo de datos debe describir la asignación de los recursos físicos a las aplicaciones de partición.
- El modelo de datos debe proporcionar suficiente información para permitir el análisis de planificabilidad de la asignación de recursos dada, así como identificar posibles optimizaciones.
- El modelo de datos debe permitir capturar la configuración del kernel de particionado: planes temporales, tablas de asignación de memoria, canales de comunicación, *health monitoring*, etc.

- El modelo de datos debe permitir la segregación del modelo completo en diferentes ficheros de forma que puedan ser utilizados por diferentes roles en el proceso de desarrollo.
- El modelo de datos debe permitir reusar datos existentes en un nuevo proyecto.
- El modelo de datos debe ser coherente y consistente con una semántica y sintaxis precisa.

Los datos del modelo se organizan de la siguiente forma:

- Modelo de partición. Representa el contrato entre el Proveedor de Aplicación (PA) y el Integrador del Sistema (IS).
    - Propiedades de la partición, tareas, puertos de comunicación, requisitos de memoria.
- Modelo físico. Describe los recursos hardware disponibles para las particiones.
    - Memoria, Procesadores, Interrupciones, periféricos.
- Modelo de sistema IMA-SP. Define la asignación de recursos a las particiones.
    - Asignación de: Memoria, Interrupciones virtuales, recursos hardware, End to end flows (ETEF).
- Modelo de kernel de particionado (PK). Describe el kernel que proporciona el particionado tanto espacial como temporal (TSP).
    - Requisitos de memoria, eventos, acciones y configuración por defecto del Health Monitoring, tanto del sistema como de las particiones;
- Modelo de datos del kernel de particionado. Contiene toda la información necesaria para configurar el PK para un proyecto dado.
    - Tabla del Health Monitoring Table, Scheduling Plan, tabla de conexiones de los puertos, tabla de configuración de la memoria.
- Modelo de datos común. Contiene los tipos básicos que utilizan el resto de modelos.

Las dependencias de cada uno de estos sub-modelos que conforman el modelos de datos total se muestran en la Figura 1.

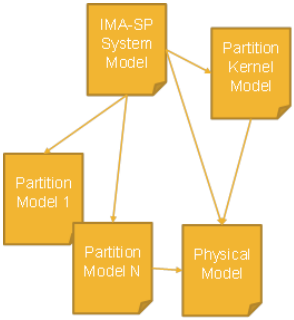


Figura 1 Dependencias del modelo de datos

Para especificar el modelo de datos IMA-SP SDT se utilizará el formato *ecore*, del framework de modelado de Eclipse (EMF, Eclipse Modeling Framework) [2]. Los ficheros del modelo están en formato XML Meta-data Interchange (XMI) [3].

4.-Flujo de trabajo IMA-SP SDT

Junto con el modelo, también se ha definido el flujo de trabajo para especificar cómo las herramientas propuestas se utilizan en el proceso de desarrollo de un sistema IMA-SP para generar el modelo de datos del sistema completo. A continuación se resume este flujo de trabajo en sus principales puntos:

1. El arquitecto del sistema (SA) define el modelo físico y se lo proporciona al Proveedor de Plataforma (PS).

2. El PS define en el modelo de kernel de particionado los siguientes elementos:

    a. Requisitos de memoria del kernel.

    b. Eventos y acciones del kernel.

3. El PS define la plataforma para los modelos de partición.

4. El PS realiza una asignación de recursos inicial en el modelo de sistema IMA-SP.

5. El PS proporciona los modelos de datos al SI.

6. El SI define el número de particiones y sus propiedades.

7. El PA o el SI define los puertos, tareas y recursos requeridos por la partición en el modelo de partición.

8. El SI define la tabla del *health monitoring*, la tabla de conexiones de los puertos, la tabla de asignación de recursos y el modelo temporal basado en ETEFs. Esto lo hace en el modelo de sistema IMA-SP y en el modelo de kernel de particionado.

9. El SI realiza un análisis de planificabilidad del modelo de datos completo y almacena la configuración de esta asignación de recursos en el modelo de datos del kernel de particionado en la parte de configuración.

10. El SI genera el fichero de configuración y el modelo redactado. El modelo redactado se utiliza para validar una sola partición. De este modo se elimina del modelo toda la información de otras particiones pero manteniendo la coherencia en el uso de recursos total.

5.-Requisitos de IMA-SP SDT

El modelo de datos descrito en el apartado 3 se ha usado como base para definir los requisitos del conjunto de herramientas IMA-SP SDT que trabaja con este modelo. Las funcionalidades mas importantes del IMA-SP SDT son:

- **Asignación de recursos**. Esta parte era un lento proceso manual realizado por el SI que debe encajar las necesidades de recursos de las particiones con el hardware disponible.

- **Análisis de planificabilidad.** En esta fase, debe validarse que la asignación de recursos anterior es realizable, es decir, que los requisitos de cada partición se cumplen con el hardware disponible. Este proceso suele ser automático.

- **Redacción.** El modelo redactado se utiliza para validar una sola partición. De este modo se elimina del modelo toda la información de otras particiones pero manteniendo la coherencia en el uso de recursos total.

- **Configuración de la plataforma.** Con el modelo completo y validado, en esta fase se crea el fichero estático de configuración para el kernel seleccionado.

6.-Diseño de IMA-SP SDT

Con el modelo de datos y los requisitos que debe cumplir el conjunto de herramientas se diseña el IMA-SP SDT.

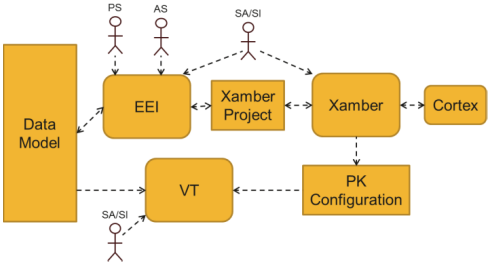


Figura 2 Arquitectura IMA-SP SDT

La arquitectura general se presenta en la Figura 2. El proceso de definición y desarrollo de un sistema IMA-SP implica los siguientes pasos y herramientas:

1. Definición inicial del modelo. Para ello se utiliza la herramienta EEI (Edit. Export and Import). Esta herramienta es un editor de EMF en la cual cada rol introduce la información inicial (definición de recursos hardware en el modelo físico por el SI, definición de requisitos de cada partición por cada PA y definición el kernel de particionado por el PS).

2. En el siguiente paso se realiza la asignación de recursos en función de las necesidades de cada aplicación y del kernel. Esto lo realiza el SI exportando el modelo emf para poder ser importado por la herramienta Xamber. Xamber será la herramienta encargada de realizar de forma automática esta asignación de recursos.

3. Xamber proporciona una representación gráfica del modelo de datos permitiendo ejecutar el planificador para generar el plan temporal y la asignación de memoria.

4. Una vez asignados todos los recursos por Xamber, el modelo de datos está completo. El SI puede entonces importar desde EEI el proyecto de Xamber para así tener el modelo completo en EMF.

5. Al mismo tiempo, el SI puede generar el fichero de configuración del sistema en Xamber. Xamber es capaz de generar este fichero para XtratuM que es el kernel de particionado seleccionado para realizar el caso de uso que validará la bondad de IMA-SP SDT.

6. Para verificar que el fichero de configuración generado se corresponde con el modelo de datos en EMF, la herramienta VT comprueba la coherencia y corrección de ambos.

Estos pasos deben seguirse de forma manual por los distintos actores del proceso de desarrollo. De todas formas, si se salta algún paso, EEI, Xamber y V son vcapaces de detectar la inconsistencia o falta de datos y avisar sin que se produzcan errores.

7.-Demostrador IMA-SP SDT

El caso de uso para demostrar la bondad y utilidad de las herramientas propuestas fue el Caso de Uso B del proyecto IMA-SP. El sistema está compuesto por 3 particiones:

- Partición de sistema: responsable de la supervisión del sistema y comunicación entre particiones e hipervisor.
- Partición MIRAS: Responsable del manejo de la instrumentación para los experimentos de la misión SMOS (Soil Moisture and Ocean Salinity).
- Partición AOCS: responsable de las aplicaciones de guiado y control del satélite.

El hypervisor utilizado es XtratuM y los requerimientos de hardware son:

- Procesador LEON3.
- Reloj del procesador a 50Mhz.
- 48 MB SDRAM.

Con el conjunto de herramientas IMA-SP SDT se definió el flujo de trabajo y así se configuró la totalidad del sistema siguiendo los siguientes pasos:

1. Definición del modelo físico por el SI con EEI.
2. Definición del modelo de plataforma por el SI con EEI.
3. Definición del modelo de kernel de particionado por el PS con EEI.
4. Definición de los modelos de partición por los AS con EEI.
5. Definición de conexiones de comunicación entre particiones por SI con EEI.
6. Asignación de recursos por el SI con Xamber.
7. Creación del fichero de configuración para XtratuM por el SI con Xamber.
8. Importación del modelo completo por el SI con EEI.

9. Verificación de la corrección y coherencia de modelo y fichero de configuración por el SI con VT.

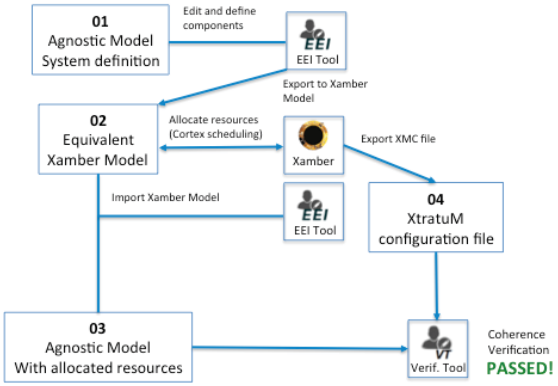


Figura 3 Flujo de trabajo IMA-SP

En la Figura 3 se muestra el flujo de trabajo descrito anteriormente. Los pasos 1 a 5 se corresponden con el recuadro `01 Agnostic Model Definition` realizado con la herramienta EEI. En la Figura 4 se puede ver el aspecto de esta herramienta.

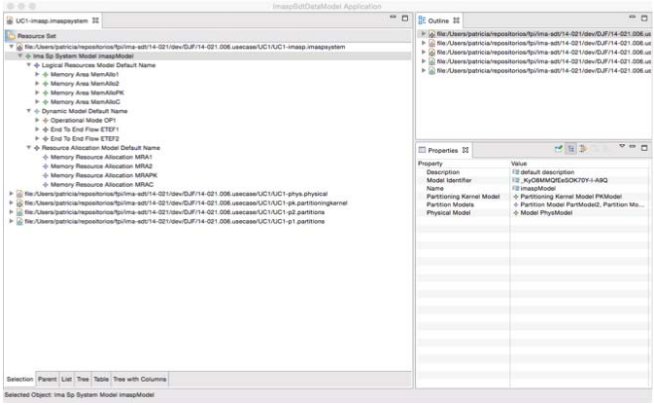


Figura 4 Herramienta EEI

Previo al paso 6, es necesario exportar el modelo de datos al formato de Xamber, por ello el recuadro '02 Equivalent Xamber Model' es la entrada a la herramienta Xamber (Figura 5).

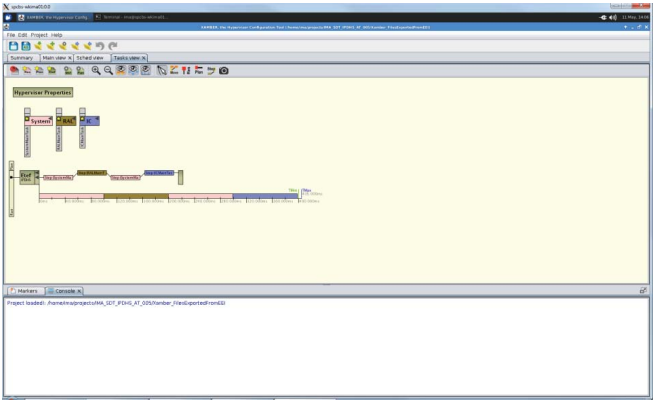


Figura 5 Herramienta Xamber

Una vez asignados todos los recursos se genera, por un lado el fichero de configuración del sistema en formato XMCF y, por otro lado, el modelo de datos de Xamber (con asignación de recursos) se importa desde la herramienta EI. Con el modelo de datos y el fichero XMCF, se comprueba mediante la herramienta VT (Figura 6) si estos dos son coherentes entre sí.

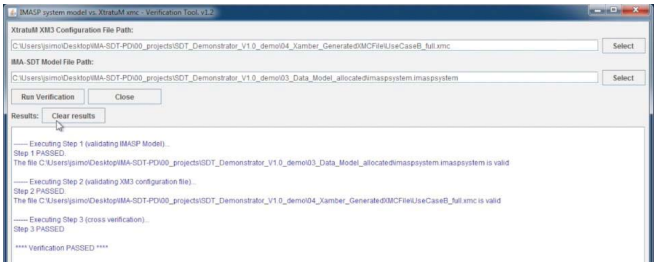


Figura 6 Herramienta VT

8.-Resultados

El resultado de utilizar el conjunto de herramientas desarrollado es la configuración del sistema (fichero XMCF y modelo de datos agnóstico completo). Por motivos de confidencialidad no se puede mostrar el contenido del fichero. Sin la existencia de las herramientas desarrolladas tendría que haberse generado la configuración a mano. Esto es perfectamente posible cuando hay pocas particiones en el caso de la asignación de memoria. Pero cuando hay muchas particiones o en el caso de la asignación de recursos temporales, es imprescindible contar con la ayuda de herramientas que automaticen el proceso, por lo que el conjunto de herramientas IMA-SDT supone un ahorro de tiempo importante en la generación de la configuración del sistema.

9.-Conclusiones y trabajo futuro

En este trabajo se ha descrito la definición, diseño e implementación de un conjunto de herramientas para la configuración de sistemas particionados que siguen la arquitectura IMA-SP. Se ha definido completamente el modelo de datos y se ha organizado siguiendo los roles definidos en el proceso de desarrollo IMA-SP. Con el desarrollo de un caso de estudio se ha demostrado su viabilidad.

Como trabajo futuro se espera ampliar el conjunto de herramientas para soportar la arquitectura multinúcleo, alinear el modelo de datos con la arquitectura OSRA (On-Board Software Reference Architecture) y soportar la configuración de otros kernels de particionado.

Bibliografía

[1] IMA-SP D08-11 – IMA development process, roles and tools, Issue 2.0, Date 21/01/2013.
[2] http://www.eclipse.org/modeling/emf/
[3] http://www.omg.org/spec/XMI/

Interpretación de dos algoritmos EDF *on-line* para la optimización de sistemas distribuidos de tiempo real

Juan M. Rivas, y J. Javier Gutiérrez

Grupo de Ingeniería Software y Tiempo-Real, Universidad de Cantabria
{rivasjm, gutierjj}@unican.es

Resumen. Los planificadores EDF (*Earliest Deadline First*) *on-line* calculan los plazos de planificación en tiempo de ejecución y se utilizan habitualmente en sistemas de tiempo real laxo. En este trabajo se propone una interpretación de dos de estos algoritmos de planificación, EQS (*Equal Slack*) y EQF (*Equal Flexibility*), para su adaptación a sistemas distribuidos de tiempo real estricto en los que la asignación de parámetros de planificación se realiza *off-line*, es decir, antes de la ejecución del sistema. La adaptación propuesta permite asignar plazos de planificación en sistemas planificados por EDF, pero también permite asignar prioridades en sistemas planificados por prioridades fijas. Los resultados obtenidos en la evaluación de los algoritmos propuestos los colocan como los más adecuados para planificar sistemas en los que los plazos son superiores a los periodos, circunstancia que suele ser habitual en los sistemas distribuidos, y secuencias largas de tareas y mensajes en respuesta a los eventos.

Palabras clave: tiempo real, sistemas distribuidos, optimización, prioridades fijas, EDF.

1 Introducción[1]

La política de planificación EDF (*Earliest Deadline First*) continúa captando la atención de investigadores en los ámbitos académico e industrial, en especial por los beneficios que se pueden obtener en el incremento del uso de los recursos procesadores. Podemos encontrar planificadores EDF en (1) lenguajes de tiempo real como Ada [1] o Java (RTSJ [2]), (2) en sistemas operativos de tiempo real como SHaRK [3], ERIKA [4], y OSEK/VDX (implementado en el nivel de aplicación) [5], (3) en redes de comunicaciones de tiempo real como el bus CAN [6] u otras de propósito general [7], o (4) en software de intermediación de tiempo real como RT-CORBA [8].

Por otra parte, en la actualidad es común encontrarnos con sistemas de tiempo real constituidos por varios procesadores conectados mediante una o varias redes de comunicación, por lo que las aplicaciones que se ejecutan en ellos tienen una naturaleza distribuida. Incluso los actuales procesadores con varios núcleos (en particular los *many-core*) permiten concebir las aplicaciones como si fueran distribuidas. En estos sistemas

[1] Este trabajo ha sido financiado en parte por el Gobierno de España en el proyecto TIN2014-56158-C4-2-P (M2C2).

se pueden tener secuencias de tareas y mensajes que se ejecutan en diferentes procesadores, o son enviados por diferentes redes de comunicaciones en respuesta a los eventos que se generan en el entorno.

Sobre los sistemas distribuidos planificados por EDF se suelen aplicar técnicas de asignación de plazos de planificación que normalmente hacen una distribución de los requisitos temporales en función de los tiempos de ejecución de peor caso [9][10][11]. En un trabajo reciente [12] demostramos que en algunos casos la aplicación de estas técnicas a EDF obtenía unos resultados muy pobres en cuanto a la utilización que se podía alcanzar en los recursos procesadores, y propusimos nuevas técnicas de asignación de plazos que rompían con la regla de que la suma de los plazos asignados a una secuencia de tareas debía cumplir con el requisito temporal establecido.

En este trabajo, nos centramos en dos algoritmos de planificación, EQS (*Equal Slack*) y EQF (*Equal Flexibility*), que fueron propuestos en [13] para la asignación *on-line* de plazos de planificación en sistemas de tiempo real laxo, y que se han seguido utilizando con este fin [14][15][16]. Estos algoritmos tampoco cumplen con la mencionada regla de suma de plazos de planificación, y el objetivo es hacer una interpretación de los mismos para obtener nuevos métodos de asignación *off-line* de parámetros de planificación, que puedan aplicarse por tanto a sistemas de tiempo real estricto. Aunque estos algoritmos están pensados para EDF, en este trabajo se estudiará también su comportamiento en la asignación a planificadores de prioridades fijas (FP, *Fixed Priority*).

El documento queda organizado de la siguiente manera. En el apartado 2 se describe el modelo de sistema distribuido utilizado en este trabajo. En el apartado 3 se hace un breve repaso de las técnicas de asignación de parámetros de planificación en sistemas distribuidos con las que se compararán los resultados. El apartado 4 trata sobre el proceso de interpretación de los algoritmos EQS y EQF de asignación de plazos de planificación para su adaptación como técnicas de asignación *off-line*. El estudio del rendimiento de estos dos algoritmos con respecto a las técnicas existentes se presenta en el apartado 5. Por último, en el apartado 6 se plantean las conclusiones de este trabajo.

2 Modelo del sistema

Se utiliza el modelo MAST [17][18] de sistemas distribuidos, que está alineado con el estándar MARTE [19] del OMG (*Object Management Group*). El sistema está compuesto por varios recursos procesadores (CPUs y redes de comunicaciones) que pueden usar planificación FP o EDF. El modelo de tareas está compuesto por flujos de principio a fin (flujos e2e) distribuidos, que son activados de manera periódica o esporádica con un intervalo mínimo entre llegadas. Cada flujo e2e Γ_i está compuesto por un conjunto N_i de actividades que pueden ser tareas ejecutando en un procesador o mensajes enviados a través de la red de comunicaciones. Cada actividad se activa cuando finaliza la ejecución de la anterior y está estáticamente asignada a un procesador o red. Con este modelo, el análisis de planificabilidad de los mensajes en las redes es similar al de las tareas en los procesadores.

La Figura 1 muestra un ejemplo de un flujo e2e sencillo con tres actividades, cada una ejecutando en un recurso procesador (PR_k) distinto. La llegada del evento externo

e_i activa el flujo e2e con su periodo T_i. Cada actividad τ_{ij} tiene asociado un tiempo de ejecución de peor caso (C_{ij}), y un tiempo de ejecución de mejor caso (C^b_{ij}). Los requisitos de tiempo que se consideran son los plazos de principio a fin, D_i, medidos desde la llegada del evento externo hasta que finaliza la última actividad del flujo e2e. Los plazos pueden ser mayores que los periodos. Cada actividad puede tener asignado un offset (Φ_{ij}), que indica la cantidad mínima de tiempo que debe transcurrir desde la llegada del evento externo hasta que la actividad pueda activarse.

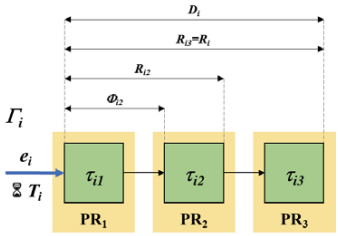


Fig. 1. Flujo de principio a fin (Γ_i) con 3 actividades

Para cada actividad τ_{ij} se define su tiempo de respuesta de peor caso (o una cota superior de éste) R_{ij} como la cantidad de tiempo máximo que tarda la actividad en ejecutarse contado desde que llega el evento externo. Similarmente se define el tiempo de respuesta de mejor caso (o una cota inferior de éste) R^b_{ij}. Estos valores de los tiempos de respuesta se obtienen tras la aplicación de una técnica de análisis de planificabilidad. Si los tiempos de respuesta de peor caso son menores o iguales que sus plazos asociados, se dice que el sistema es planificable.

Debido a las relaciones de precedencia en los flujos e2e, las activaciones pueden tener *jitter* de activación (J_{ij}), que es la variación máxima que puede experimentar la actividad τ_{ij} entre sus instantes de activación. Similarmente, el evento externo puede tener *jitter*.

Cada actividad se planifica utilizando su parámetro de planificación, que puede ser una prioridad fija (P_{ij}) para planificación FP, o un plazo de planificación para planificación EDF. Los plazos de planificación son valores utilizados para planificar, pero no representan un requisito temporal. Dependiendo de si el sistema distribuido tiene sus relojes sincronizados o no, distinguimos dos tipos de plazos de planificación:

- Plazos globales de planificación (SD_{ij}), que toman como referencia el instante de activación del flujo e2e. Hacer uso de este tipo de plazos de planificación requiere sincronización en los relojes, y por ello a este tipo de planificación lo llamamos GC-EDF (*Global-Clock* EDF).
- Plazos locales de planificación (Sd_{ij}), que toman como referencia la propia activación de la actividad. Este esquema no requiere sincronización en los relojes, y por lo tanto lo llamamos planificación LC-EDF (*Local-Clock* EDF).

Además del modelo, MAST también proporciona un conjunto de herramientas de análisis y optimización de sistemas distribuidos [20]. Las técnicas de análisis determinan la planificabilidad de los sistemas calculando tiempos de respuesta de peor caso, que se comparan con los plazos impuestos. Para sistemas distribuidos, en MAST se implementa el análisis holístico para FP [21], GC-EDF [22] y LC-EDF [10]. En MAST también se ofrece una variedad de análisis basados en *offsets* que reducen el pesimismo del análisis holístico, y que se implementan para planificación FP [23][24][25], GC-EDF [26] y LC-EDF [27]. Por último, en MAST también se implementan diferentes técnicas de asignación de parámetros de planificación que serán utilizadas en este trabajo para la evaluación de los algoritmos propuestos. Las técnicas de asignación disponibles se tratan en el siguiente apartado.

3 Asignación de parámetros de planificación

Las técnicas de asignación de parámetros de planificación son de vital importancia en el desarrollo de sistemas de tiempo real. Debido al elevado número de combinaciones de actividades y parámetros de planificación que pueden probarse en un sistema distribuido, es trivial comprobar que encontrar la asignación óptima es un problema NP-difícil [28][29]. Por lo tanto, se hace necesario el uso de técnicas no óptimas, pero que sean tratables computacionalmente.

Para la asignación de prioridades fijas se propuso el algoritmo HOPA [30] (*Heuristic Optimized Priority Assignment*), que es un algoritmo heurístico que hace uso de los tiempos de respuesta de peor caso para asignar y optimizar prioridades fijas en sistemas distribuidos. HOPA es capaz de planificar un mayor número de sistemas, y en menos tiempo, que otros algoritmos de propósito general como el templado simulado [30].

En cuanto a sistemas EDF, Liu [9] destaca los siguientes algoritmos de reparto del plazo de principio a fin: UD, en el que los plazos de planificación son iguales a los de principio a fin; ED, en el que los plazos de planificación son iguales a los de principio a fin, pero se restan los tiempos de ejecución de peor caso de las actividades posteriores en el flujo e2e; PD, en el que los plazos de principio a fin se reparten de manera proporcional a los tiempos de ejecución de cada actividad; y NPD, que es similar a PD, pero en el que además se tiende a dar plazos más largos en aquellos procesadores más cargados.

Con el objetivo de mejorar la capacidad de planificación de estos algoritmos de reparto, y utilizando como base el algoritmo HOPA, se definió el algoritmo HOSDA para la asignación de plazos de planificación en sistemas GC-EDF [31] y LC-EDF [10]. Al igual que HOPA, utiliza los tiempos de respuesta de peor caso obtenidos por alguna técnica de análisis de planificabilidad para asignar y optimizar plazos de planificación.

Estas técnicas que se han destacado están diseñadas para trabajar únicamente en sistemas distribuidos con una única política de planificación. Para superar esta limitación, se propuso el algoritmo heurístico HOSPA [11], que es una fusión entre HOPA y HOSDA para asignar tanto prioridades fijas como plazos de planificación en sistemas distribuidos en los que conviven diferentes políticas de planificación (sistemas heterogéneos). En [11] además se introduce el concepto de plazo virtual, que es un número

intermedio que se transformará en el parámetro de planificación correspondiente: prioridad fija para FP, o plazo de planificación para EDF. Este plazo virtual se puede utilizar para aplicar los algoritmos de reparto de plazos en sistemas con políticas para las cuales no fueron diseñados. HOSPA será uno de los algoritmos utilizados en este trabajo como referencia para la evaluación de las nuevas propuestas.

4 Interpretación de los algoritmos EQS y EQF para la asignación *off-line* de parámetros de planificación

El objetivo es interpretar los algoritmos EQS (*Equal Slack*) y EQF (*Equal Flexibility*) propuestos en [13] para que puedan producir plazos virtuales fijos que luego se puedan transformar en plazos de planificación locales, globales o en prioridades fijas. Así pues, como primer paso estudiamos la formulación original de estos algoritmos, que producen plazos globales de planificación, y la adaptamos al modelo de flujo e2e que utilizamos.

El algoritmo EQS asigna los plazos dividiendo equitativamente el *slack* entre las actividades del flujo e2e, entendiendo *slack* como la diferencia entre el plazo y el tiempo de respuesta de peor caso. Como *a priori* no se conocen los tiempos de respuesta, utiliza los tiempos de ejecución de peor caso como una estimación optimista de este tiempo de respuesta de peor caso. La ecuación de EQS en su formulación original se muestra a continuación [13]:

$$dl(T_i) = ar(T_i) + pex(T_i) + \frac{dl(T) - ar(T_i) - \sum_{j=i}^{m} pex(T_j)}{m - i + 1} \quad (1)$$

donde los términos en los que se expresa tienen los siguientes significados de acuerdo a los conceptos definidos en nuestro modelo:

- $dl(T_i)$: plazo absoluto asignado a la actividad *i*-ésima del flujo e2e.
- $ar(T_i)$: instante de activación de la actividad *i*-ésima del flujo e2e.
- $pex(T_i)$: tiempo de ejecución de peor caso de la actividad *i*-ésima del flujo e2e.
- $dl(T)$: plazo absoluto de principio a fin del flujo e2e.
- m: número de actividades en el flujo e2e.

El algoritmo EQS original descrito en la ecuación (1) está pensado para su ejecución *on-line*, y asigna plazos globales a las actividades teniendo en cuenta los instantes de activación de éstas. Sin embargo, en nuestro modelo pensado para herramientas de asignación *off-line*, estos instantes de activación de las actividades se desconocen. Para adaptar la ecuación (1) a la formulación del modelo MAST que usamos, asumimos que los instantes de activación de las actividades ocurren en el instante cero, lo que significa que los plazos virtuales (*Vd*) obtenidos toman como referencia la activación del flujo e2e (plazos globales de planificación). Así, el algoritmo adaptado se muestra en la siguiente ecuación:

$$Vd_{ij} = C_{ij} + \frac{D_i - \sum_{k=j}^{N_i} C_{ik}}{N_i - j + 1} \quad (2)$$

El algoritmo EQF original también se basa en el reparto del *slack* (definido de la misma manera que en EQS), pero lo realiza de manera proporcional a los tiempos de ejecución de cada actividad. Para ello define un nuevo concepto llamado "flexibilidad" (*flexibility*), que es la ratio entre el *slack* y el tiempo de respuesta (o una estimación de éste). La ecuación de EQF en su formulación original se describe a continuación [13]:

$$dl(T_i) = ar(T_i) + pex(T_i) + \left[dl(T) - ar(T_i) - \sum_{j=i}^{m} pex(T_j)\right] * \left[\frac{pex(T_i)}{\sum_{j=i}^{m} pex(T_j)}\right] \quad (3)$$

donde los términos en los que se expresa son los utilizados en EQS.

Adaptamos la ecuación original de EQF para la asignación de plazos virtuales de planificación definidos de acuerdo con nuestro modelo. Para ello seguimos el mismo criterio que aplicamos a EQS, asumiendo la activación de las actividades en el instante cero. La formulación de EQF adaptada al modelo MAST de flujo e2e se muestra en la siguiente ecuación:

$$Vd_{ij} = C_{ij} + \left[D_i - \sum_{k=j}^{N_i} C_{ik}\right] \left[\frac{C_{ij}}{\sum_{k=j}^{N_i} C_{ik}}\right] \quad (4)$$

En realidad, las adaptaciones que proponemos de EQS y EQF obtienen plazos virtuales de planificación que razonablemente sólo estarían indicados para su uso con planificación GC-EDF. Así pues, es necesaria una interpretación de estos plazos virtuales para su transformación en parámetros de planificación [11]. Para planificación FP, los plazos virtuales se transforman en prioridades fijas utilizando el criterio *Deadline Monotonic* en cada recurso procesador. Para planificación GC-EDF y LC-EDF, los plazos virtuales se usan directamente como plazos de planificación. En el caso de LC-EDF, se están utilizando plazos locales de planificación que incumplen la regla mencionada en la introducción relativa a que la suma de los plazos locales de planificación en un flujo e2e cumplan con el requisito temporal impuesto. Como se demuestra en [12], esta regla que parece muy razonable no es en absoluto necesaria, e incluso tiene consecuencias negativas en el rendimiento que se puede obtener con la planificación LC-EDF.

5 Evaluación

En este apartado, evaluamos el rendimiento de la interpretación de EQS y EQF propuesta cuando se aplica en sistemas distribuidos de tiempo real cuya planificación se realiza *off-line*. Para ello, se hace una comparación con la aplicación de las técnicas de asignación que hemos introducido en el apartado 3, para las diferentes políticas de planificación que tratamos en el apartado 2: FP, GC-EDF y LC-EDF. En esta evaluación utilizamos la herramienta GEN4MAST [32], que permite generar un conjunto de pruebas lo suficientemente amplio para obtener resultados estadísticamente relevantes, y permite además la aplicación automática de las técnicas de asignación bajo estudio.

Las características del conjunto de sistemas sintéticos generado se plantean con dos objetivos principales: (1) abarcar un conjunto amplio de sistemas lo suficientemente complejos como para dar validez a los resultados, y (2) que el estudio requiera un tiempo total de ejecución razonable. Bajo estas dos premisas, todos los sistemas generados poseen las siguientes características básicas:

- Poseen 10 flujos e2e y 5 recursos procesadores.
- Los flujos e2e no recorren el mismo recurso procesador en más de una ocasión. Esto sólo es posible si el número de actividades en el flujo e2e es menor o igual que el número de recursos procesadores. Cuando esta condición no se cumple, la localización de las actividades se realiza de forma aleatoria.
- Los periodos se seleccionan de forma aleatoria en el rango [100,1000] utilizando una distribución de probabilidad logarítmica-uniforme [33].
- Los tiempos de ejecución de peor caso de las actividades se calculan con el algoritmo *UUnifast* [34]. Los tiempos de ejecución de mejor caso son iguales a cero. Todos los recursos procesadores tienen la misma utilización.

A partir de estas características básicas, generamos el conjunto de sistemas de estudio variando otras tres características fundamentales de éstos:

- Plazos de principio a fin: se generan sistemas de forma que la ratio D_i/T_i vaya tomando todos los valores en el conjunto {1,2,4,6,8,10,12,14,16,18,20}. Estas variaciones se hacen con sistemas con 10 actividades por flujo e2e.
- Número de actividades en los flujos e2e con valores en el conjunto {4,6,8,10,12,14,16,18,20}. Estas variaciones se hacen con plazos de principio a fin $D_i=N_i*T_i$ para sistemas FP y LC-EDF, y $D_i=5*T_i$ para sistemas GC-EDF, donde N_i y T_i son respectivamente el número de actividades y el periodo de los flujos e2e.
- Series de utilizaciones medias del sistema en el rango [10, 96](%), con variación del 1%. Esta generación de series se realiza para todos los sistemas.

Para obtener resultados estadísticamente relevantes, se generan 30 sistemas con cada una de las combinaciones de las características evaluadas. Una vez generado el conjunto de sistemas, se aplican sobre éste las técnicas de asignación de parámetros de planificación que se han citado previamente: UD, ED, PD, EQS, EQF y HOSPA. Estudiaremos por separado las tres políticas de planificación (FP, LC-EDF y GC-EDF) y en todos los casos utilizaremos el análisis holístico [21][22][10] para determinar la planificabilidad del sistema. Cómo métrica para comparar el rendimiento de las diferentes técnicas, nos centraremos en observar cuál es la utilización máxima planificable (UMP) alcanzada por cada una de las técnicas de asignación utilizada. Una UMP mayor indica que la asignación de parámetros de planificación fue capaz de encontrar una solución planificable para sistemas con mayor carga computacional.

En la Figura 2 se muestra la UMP media obtenida por las diferentes técnicas cuando se utilizan para asignar prioridades en planificación FP. En el estudio de la UMP en función de los plazos de principio a fin (Figura 2a) observamos dos zonas diferenciadas. En primer lugar, para plazos de principio a fin pequeños ($D_i=T_i$), las mejores técnicas son HOSPA y PD, que consiguieron planificar sistemas que en promedio tenían un 30%

de utilización. En esta zona, EQS, EQF, ED y UD obtienen un rendimiento claramente inferior, con alrededor de un 20% de UMP media. Por otro lado, a medida que los plazos de principio a fin aumentan, la UMP media de EQS y EQF crece a un ritmo superior que el del resto de técnicas. Observamos que a partir de aproximadamente D_i=4*T_i, EQF adelanta a HOSPA, convirtiéndose en la técnica que mayores UMP medias obtiene.

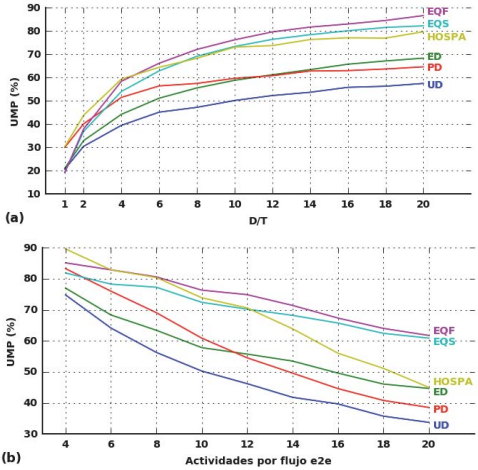


Fig. 2. UMP media (%) para sistemas planificados con FP: (a) diferentes plazos de principio a fin; (b) diferentes números de actividades por flujo e2e.

Si observamos los resultados para sistemas con diferentes números de actividades por flujo e2e (Figura 2b**)**, también se distinguen dos zonas. Mientras que para 4 actividades/flujo e2e el algoritmo HOSPA es el que mayor UMP media obtuvo (90%), a partir de 6 actividades/flujo EQF adelanta a HOSPA en términos de UMP media. La diferencia de rendimiento reportada entre EQF y HOSPA aumenta a medida que los flujos e2e son más largos. Para flujos e2e con 20 actividades, EQF posee una UMP media 17% superior a la obtenida por HOSPA.

En la Figura 3 se muestran los resultados de las diferentes técnicas bajo estudio cuando se utilizan para asignar plazos globales de planificación para GC-EDF. En este caso, y a diferencia de lo observado previamente para FP, se comprueba que el rendimiento de EQS y EQF se ve claramente superado por HOSPA, tanto para diferentes plazos de principio a fin (Figura 3a), como de números de actividades por flujo e2e (Figura 3b). Cabe notar que para los resultados de la Figura 3b se está utilizando un

plazo de principio a fin D_i=5*T_i. Esta modificación se realiza a la vista del rendimiento de HOSPA y PD para plazos de principio a fin superiores a D_i=4*T_i (ver Figura 3a). en el que consiguen planificar sistemas con hasta el 96% de utilización (la máxima generada). Este rendimiento se asemeja al de la planificación EDF en sistemas monoprocesadores y tareas independientes.

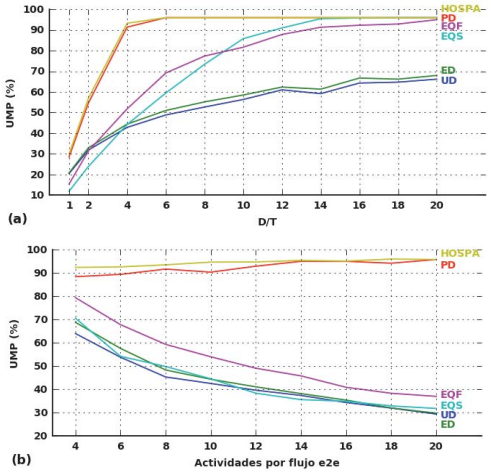


Fig. 3. UMP media (%) para sistemas planificados con GC-EDF: (a) diferentes plazos de principio a fin; (b) diferentes números de actividades por flujo e2e.

Por último, en la Figura 4 se evalúa el rendimiento de las diferentes técnicas bajo estudio cuando asignan plazos locales de planificación para LC-EDF. En este caso, los algoritmos HOSPA y PD se modifican utilizando el criterio LC-EDF-GSD propuesto en [12] para asignar plazos locales de planificación con valores equivalentes a plazos globales. El motivo es comparar las nuevas propuestas con las que obtienen los mejores resultados, que como se demuestra en [12] consiste en el uso de este criterio para la asignación de plazos locales de planificación en LC-EDF. A la vista de los resultados mostrados en la figura, y al contrario de lo observado para planificación GC-EDF, el algoritmo EQF aquí posee un rendimiento que en promedio supera al de las técnicas existentes, aventajando a HOSPA en hasta un 7% en términos de UMP media. La superioridad de EQF en UMP media se mantiene en todos los casos estudiados, salvo para plazos de principio a fin iguales a los periodos (Figura 4a), en el que todas las técnicas obtienen una UMP media en torno al 20%.

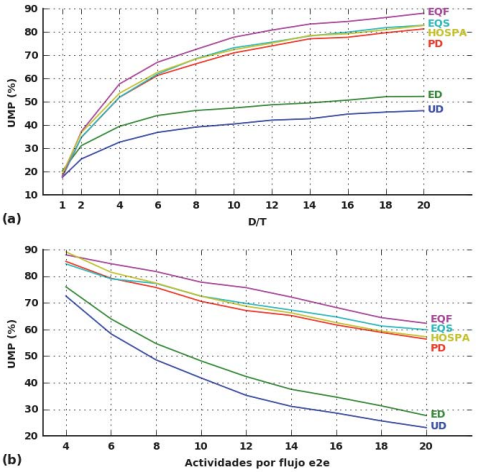


Fig. 4. UMP media (%) para sistemas planificados con LC-EDF: (a) diferentes plazos de principio a fin; (b) diferentes números de actividades por flujo e2e.

6 Conclusiones

En este trabajo se ha presentado la adaptación como técnica *off-line* de dos algoritmos de asignación de plazos de planificación, EQS y EQF, que en su origen fueron diseñados para la planificación EDF *on-line* de sistemas de tiempo real laxo. Esta adaptación permite su aplicación a sistemas distribuidos de tiempo real estricto planificados por prioridades fijas, o EDF tanto con relojes locales como con reloj global. En todos los casos la planificabilidad de los sistemas resultantes se puede comprobar mediante la aplicación de las técnicas de cálculo de tiempos de respuesta correspondientes.

Se ha evaluado el rendimiento de las adaptaciones propuestas de ambos algoritmos, comparándolos con las técnicas de asignación de parámetros de planificación existentes. Como resultado de esta evaluación, se ha comprobado cómo EQF, y en menor medida también EQS, poseen una capacidad de planificación equiparable, y en muchas situaciones superior, al algoritmo heurístico HOSPA para sistemas con planificación FP y LC-EDF. De los resultados obtenidos cabe destacar lo siguiente:

- Es sorprendente que estos algoritmos obtengan los peores resultados para la planificación GC-EDF que es para la que fueron diseñados como técnicas *on-line*.

- Se dispone de dos nuevos algoritmos con los que optimizar la asignación de prioridades en FP, o de plazos locales de planificación en LC-EDF, para sistemas distribuidos con plazos superiores al periodo o con flujos e2e largos.
- Los algoritmos propuestos no son iterativos, por lo que sus resultados se obtienen muy rápidamente en comparación con cualquier otra opción iterativa como HOSPA.

Como conclusión general, se observa que en la asignación de parámetros de planificación ninguno de los algoritmos es el mejor en todas las situaciones. Por este motivo, en la optimización de un sistema, quizá la solución pase por probar todos los algoritmos que no son iterativos, y si no se encuentra una solución, probar entonces los iterativos con las mejores soluciones encontradas por los anteriores como asignación inicial.

7 Bibliografía

1. ISO/IEC, 2012. Ada 2012 Reference Manual. Language and Standard Libraries - International Standard ISO/IEC 8652:2012(E) (2012).
2. RTSJ (Real-Time Specification for Java) home page, http://www.rtsj.org
3. S.Ha.R.K. (Soft Hard Real-Time Kernel) home page, http://shark.sssup.it/
4. ERIKA Enterprise, Evidence home page, http://www.evidence.eu.com/
5. C. Diederichs, U. Margull, F. Slomka, and G. Wirrer, “An application-based EDF scheduler for OSEK/VDX". Design, Automation and Test in Europe, DATE '08 , págs. 1045-1050 (2008)
6. P. Pedreiras, and L. Almeida, “EDF message scheduling on controller area network”. Computing & Control Engineering Journal 13(4), págs. 163-170 (2002).
7. M. Di Natale, and A. Meschi, “Scheduling Messages with Earliest Deadline Techniques”. Real-Time Systems 20(3), págs. 255-285 (2001).
8. OMG (Object Management Group), Realtime Corba Specification. v1.2, 2005. http://www.omg.org/spec/RT/1.2/
9. J.W.S. Liu, "Real-time systems". Prentice Hall (2000).
10. J.M. Rivas, J.J. Gutiérrez, J.C. Palencia and M. González Harbour, "Optimized Deadline Assignment and Schedulability Analysis for Distributed Real-Time Systems with Local EDF Scheduling". Proc. of the 8th International Conference on Embedded Systems and Applications (ESA), Las Vegas (Nevada, USA), págs. 150-156 (2010).
11. J.M. Rivas, J.J. Gutiérrez, J.C. Palencia, Michael González Harbour, “Schedulability analysis and optimization of heterogeneous EDF and FP distributed real-time systems”. Proc. of the 23rd Euromicro Conference on Real-Time Systems, Porto, págs. 195-204 (2011).
12. J.M. Rivas, J.J. Gutiérrez, J.C. Palencia and M. González Harbour, “Deadline Assignment in EDF Schedulers for Real-Time Distributed Systems". IEEE Transactions on Parallel and Distributed Systems, Vol. 26(10), págs. 2671-2684 (2015).
13. B. Kao and H. Garcia-Molina, "Deadline Assignment in a Distributed Soft Real-Time System". IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 12 (1997).
14. J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing". Journal of Grid Computing, vol. 3, no. 3-4, págs. 171-200 (2005).
15. Z. Wu, X. Liu, Z. Ni, D. Yuan and Y. Yang, "A market-oriented hierarchical scheduling strategy in cloud workflow systems". The Journal of Supercomputing, vol. 63, no. 1, págs. 256-293 (2013).

16. J.H. Son and M. Ho Kim, "Improving the performance of time-constrained workflow processing". Journal of Systems and Software, vol. 58, no. 3, págs. 211-219 (2001).
17. Michael González Harbour, J.J. Gutiérrez, J.C. Palencia, J.M. Drake, "MAST: Modeling and Analysis Suite for Real Time Applications". Proc. of the 13th Euromicro Conference on Real-Time Systems, Delft (The Netherlands), págs. 125-134 (2001).
18. Michael González Harbour, J.J. Gutiérrez, J.M. Drake, P. López, J.C. Palencia, "Modeling distributed real-time systems with MAST 2". Journal of Systems Architecture, vol 56, no. 6, Elsevier, págs. 331-340 (2013).
19. Object Management Group, "UML profile for MARTE: Modeling and Analysis of Real Time Embedded Systems, version 1.1". OMG document formal/2011-06-02 (2011).
20. MAST, página web http://www.mast.unican.es
21. K.W. Tindell and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems". Microprocessing and Microprogramming, vol. 50, no. 2-3 (1994).
22. M. Spuri, "Analysis of Deadline Scheduled Real-Time Systems". Research Report RR-772, INRIA, France (1996).
23. K.W. Tindell, "Adding Time-Offsets to Schedulability Analysis". Department of Computer Science, University of York, Technical Report YCS-221 (1994).
24. J. Mäki-Turja and M. Nolin. "Efficient implementation of tight response-times for tasks with offsets". Real-Time Systems Journal 40(1), págs. 77-116 (2008).
25. J.C. Palencia and M. González Harbour, "Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems". Proc. of the 20th Real-Time Systems Symposium (RTSS), págs. 328-339 (1999).
26. J.C. Palencia and M. González Harbour, "Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF". Proc. of the 15th Euromicro Conference on Real-Time Systems (ECRTS), Porto (Portugal), págs. 3-12 (2003).
27. U. Díaz-de-Cerio, J.P. Uribe, M. González Harbour and J.C. Palencia, "Adding precedence relations to the response-time analysis of EDF distributed real-time systems". Proc. of the 22nd International Conference on Real-Time Networks and Systems, págs. 129-138 (2014).
28. A. Burns, "Scheduling Hard Real-Time Systems: A Review". Software Engineering Journal vol. 6, no. 3, págs. 116-128 (1991).
29. K.W. Tindell, A. Burns and A.J. Wellings, "Allocating Real-Time Tasks. An NP-Hard Problem Made Easy". Real-Time Systems Journal, vol. 4, no. 2, págs. 145-165 (1992).
30. J.J. Gutiérrez and M. González Harbour, "Optimized priority assignment for tasks and messages in distributed hard real-time systems". Proc of the 3rd Third Workshop on Parallel and Distributed Real-Time Systems (WPDRTS), Santa Barbara (California, USA), págs. 124-132 (1995).
31. J.M. Rivas and J.J. Gutiérrez (Director), "Algoritmo de asignación de plazos globales en sistemas distribuidos de tiempo real con planificación EDF: comparativa de estrategias de planificación". Tesis de Máster, Universidad de Cantabria (2009).
32. J.M. Rivas, J.J. Gutiérrez and M. González Harbour, "GEN4MAST: A Tool for the Evaluation of Real- Time Techniques Using a Supercomputer". Proc. of the 3rd International Workshop on Real-Time and Distributed Computing in Emerging Applications (REACTION) (2014)
33. P. Emberson, R. Stafford and R.I. Davis, "Techniques for the synthesis of multiprocessor tasksets". Proc. of the 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), Brussels, págs. 6-11, (2010).
34. E. Bini and G.C. Buttazzo, "Measuring the performance of schedulability tests". Real-Time Systems, vol 30, no. 1-2, págs. 129-154 (2005).

AQUILAFUENTE,223

Con este mensaje os damos la bienvenida a la V edición del Simposio de Sistemas de Tiempo Real que se celebra en el marco del Congreso Español de Informática, CEDI 2016. Este evento, que se lleva organizando desde la primera edición del CEDI, es una oportunidad para que la comunidad española de sistemas de tiempo real pueda dar visibilidad a sus últimos trabajos en un entorno más amplio y heterogéneo que el de las Jornadas de Tiempo Real que se vienen organizando anualmente de manera ininterrumpida desde 1998, y en este año 2016 han cumplido su XIX edición.

En esta ocasión, tenemos la fortuna de abrir el programa con la intervención de Francisco J. Cazorla, investigador del CSIC, que impartirá una charla invitada sobre uno de los temas que más interés y oportunidades de investigación está despertando en el área de tiempo real: los procesadores multicore. Francisco J. Cazorla es investigador del CSIC en el BSC (*Barcelona Supercomputing Center*), donde dirige el grupo CAOS (*Computer Architecture Operating System Interface*). Su trabajo se centra principalmente en sistemas computadores con énfasis en el diseño de hardware para tiempo real y alto rendimiento, y en técnicas de análisis temporal.

El programa técnico cuenta con nueve trabajos de gran calidad que han sido revisados por el Comité de Programa, y que aparecen en estas actas agrupados por los siguientes temas: (1) middleware de comunicaciones y casos de estudio, (2) sistemas operativos y gestión de recursos, y (3) modelado, análisis temporal, configuración y optimización.

Desde la organización nos gustaría expresar nuestro agradecimiento al conferenciante invitado y a todas las personas que con sus presentaciones han contribuido al programa del simposio. También queremos agradecer a todos los participantes su presencia y aportaciones, así como a la organización local por facilitar la celebración del evento. Esperamos que el programa sea de vuestro agrado.

Un afectuoso saludo del Comité Organizador del V Simposio de Sistemas de Tiempo Real

Entidades colaboradoras:

SpAin Ada

UC UNIVERSIDAD DE CANTABRIA

SCIE SOCIEDAD CIENTÍFICA INFORMÁTICA DE ESPAÑA

VNiVERSiDAD D SALAMANCA CAMPUS DE EXCELENCIA INTERNACIONAL

GRUPO DE INVESTIGACIÓN BISITE .usal.es