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*).

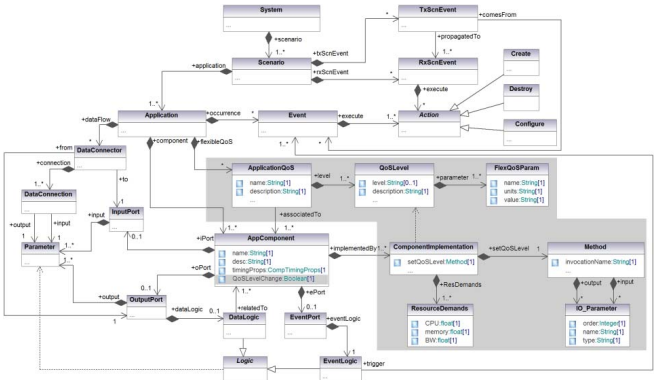
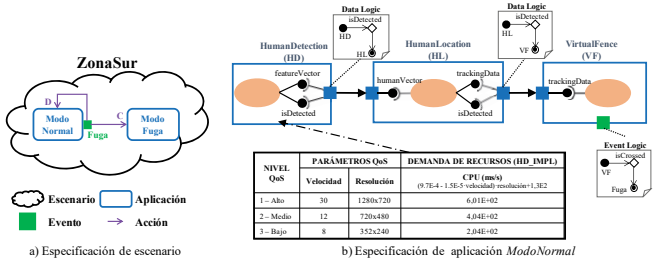


Fig. 1. Meta-Modelo para la especificación de aplicaciones

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.

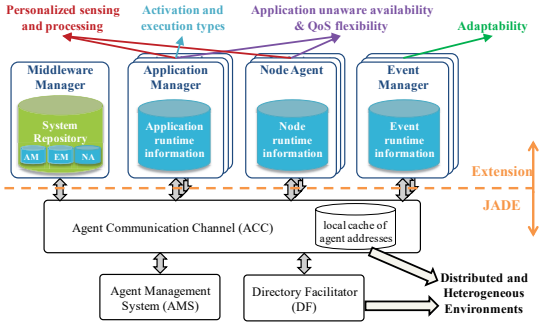


Fig. 3. Arquitectura de middleware basada en multi-agentes (MAS-RECON)

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)

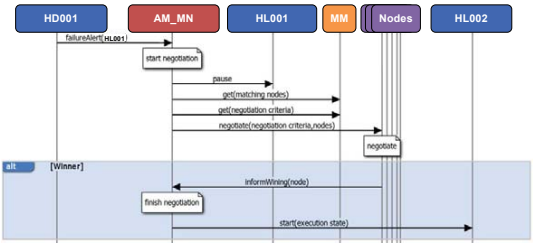


Fig. 4. Detección de fallo y recuperación con estado en la aplicación *ModoNormal*

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*).

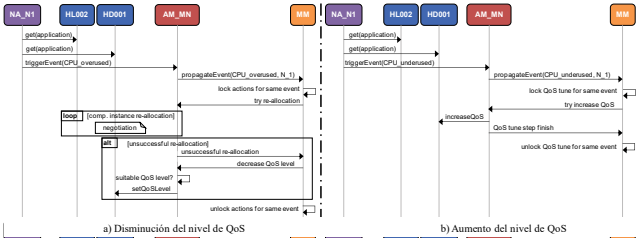


Fig. 5. Demanda flexible de recursos en aplicaciones multimedia

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.

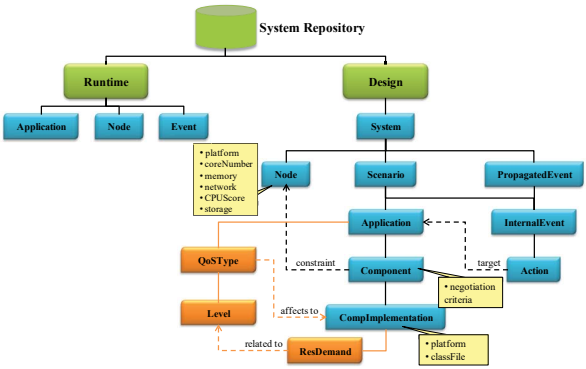


Fig. 6. Estructura extendida del System Repository

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).

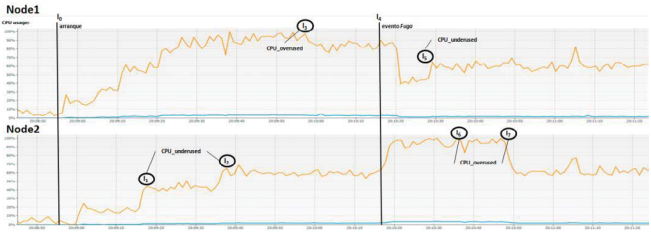


Fig. 7. Gestión de la flexibilidad de QoS del nivel de aplicación con respecto al uso de CPU

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)

