eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Departamento Ingeniería de Sistemas y Automática

Escuela de Ingeniería de Bilbao

INGENIERÍA BASADA EN MODELOS APLICADA A SISTEMAS DISTRIBUIDOS SENSIBLES AL CONTEXTO

TESIS DOCTORAL

Dña. Aintzane Armentia Díaz de Tuesta

Directoras: Marga Marcos Muñoz

Elisabet Estévez Estévez

Bilbao, Octubre de 2016

Oinatzi,

denbora bera baino lan hau egiten emandako uneak zor baitizkiot.

Hace unos cinco años inicié un doble proyecto, uno personal y otro laboral. El primero no para de crecer, y el segundo está a punto terminar, por lo que a lo largo de estos párrafos quisiera agradecer a todas esas personas que de una u otra manera me han ayudado a conseguirlo.

En primer lugar a mis directoras de tesis, Marga y Eli. Vuestra guía y ayuda ha sido indispensable para hacer posible este trabajo. Muchas gracias por todo el tiempo y apoyo que me habéis brindado, especialmente en los últimos meses. No ha sido fácil, por momentos me ha parecido hasta imposible, pero gracias a vuestra confianza y ánimos lo he logrado.

Quisiera agradecer también el apoyo que he recibido de todos mis compañeros y compañeras del Departamento de Ingeniería de Sistemas y Automática. Nagore, zu bezalako pertsona bat ezagutzea benetazko ohorea izan da. Eskerrik asko beti hor egotegatik. Itzi, incansables en nuestras tertulias arreglando el mundo, nunca me habían dado tantos besos y abrazos en el trabajo. No lo cambies. Eva, por formar parte de la terapia de grupo del comedor donde tantas veces nos hemos desahogado. Marga, por todo lo anterior y por tus consejos sobre la vida cuando más lo he necesitado. Gracias también a "las chicas", también conocidas como Arantza, Isabel y Mari Luz, siempre preparadas para ayudar "en lo que sea, de verdad", tú diles que allí van ellas. No me quiero olvidar de las que están un poco más lejos. Eli, que aunque te fuiste un poco al sur, buscando el calor, siempre estás ahí, dándolo todo. Gracias por acogerme en tu casa y hacerme un hueco en vuestras vidas.. Edurne, muchas gracias por los ánimos que siempre me has dado. Nunca olvido aquel viaje a Valencia y la cantidad de gente que conocimos. Y la más lejana, Lupita, qué padre haberte conocido. Te echo mucho de menos. Y aunque somos muchas chicas, también hay chicos. Gracias a Darío, Fede y

Asier por estar siempre disponibles y dispuestos a echar un cable. A Rafa y Unai, compañeros de tesis, cacharreros hasta la médula. Aunque la universidad considere que vuestras prácticas están al margen de la ley, gracias a vosotros el demostrador ha sido posible. Hacen falta más palmeritas para animar el trabajo en el laboratorio.

No me quiero olvidar de la gente del Grupo de Robótica, Automática y Visión por Computador de la Universidad de Jaén. Aparte de la experiencia laboral, con vosotros he pasado muy buenos ratos y me he sentido realmente a gusto. Muchas gracias.

Fuera del trabajo también he podido contar con gente excepcional. Gracias a la kuadri por todos vuestros ánimos y por creer en mí. Ya he pasado más años con vosotras que sin vosotras, y eso se nota. Gracias a Cris, Rakel y Bea; y en especial a Sonia, Ainhoa y Nekane, porque me habéis acompañado en los momentos más duros, cuando todo se veía de color negro. Gracias por llamar y por escuchar. Por no juzgar. Tenemos que ir a celebrarlo a lo loco, podemos empezar por volver al centro gallego. A ver si los ribeiros siguen estando tan ricos como antes. Quiero agradecer el haber tenido la oportunidad de conocer a unas telekos muy lokas, Amaia, Sara, Vanessa, Marce y Ainara, con las que he aprendido que otro estilo de vida es posible. Un café en Bilbao no arregla el mundo, pero sienta de maravilla. Gracias a todas vosotras.

Por último quiero agradecer todo el apoyo de mi familia: aita, ama, Iván y Ainhoa. Sois el equipo A, la patrulla canina a vuestro lado son unos aficionados. Siempre preparados para el rescate. Gracias por todo el tiempo que me habéis dedicado para que pueda acabar este trabajo. Gracias por vuestro apoyo y dedicación, porque sin ellos hoy no estaría aquí. Oinatz, se acabó ese "gran" trabajo, a partir de ahora tendré más tiempo para ti.

RESUMEN

En esta Tesis Doctoral se plantea una metodología, soportada por mecanismos y herramientas, que da soporte al *ciclo de desarrollo* de *aplicaciones distribuidas sensibles al contexto*, aquéllas que supervisan su entorno físico con objeto de detectar cambios en él y reaccionar rápida y adecuadamente. Se trata de aplicaciones presentes en diferentes campos de aplicación que demandan requisitos tales como ejecución en entornos distribuidos y heterogéneos, personalización de las tareas de supervisión, adaptación a cambios relevantes en su contexto, gestión de la calidad específica de cada ámbito o aplicación, disponibilidad y recuperación ante situaciones de fallo.

En concreto, se propone una **aproximación de modelado** genérica que permite capturar la especificación y diseño de estas aplicaciones, independientemente de la plataforma de gestión responsable de su ejecución y atendiendo a los diferentes expertos que participan: expertos de dominio y desarrolladores de software. Se hace uso de la ingeniería dirigida por modelos para lograr la separación de dominios necesaria, de manera que el *experto de dominio* realiza el diseño arquitectónico de las aplicaciones en el que se especifican todos sus requisitos, mientras que el *desarrollador de software* se centra en el diseño e implementación de la solución software correspondiente. Por lo tanto, la aproximación de modelado recoge los requisitos de las aplicaciones que una plataforma de gestión debe cumplir en tiempo de ejecución, al mismo tiempo que captura toda la información necesaria para la generación del código de dichas aplicaciones.

También se plantea un **entorno de desarrollo integrado** (IDE) que, basado en dicha aproximación de modelado, da soporte al ciclo de desarrollo de las aplicaciones. Se ha desarrollado un prototipo del IDE que se ha validado con la especificación y desarrollo de un demostrador en el campo de la asistencia domiciliaria.

ABSTRACT

This research work presents a methodology, supported by mechanisms and tools, that *covers the development cycle of distributed context-aware applications*, those that supervise their physical environment in order detect relevant changes and provide a rapid and suitable reaction. These applications belong to different domains, demanding requirements such as distributed and heterogeneous environments, customization of supervision tasks, evolution according to changes on their context, management of the application specific quality, availability and failure recovery.

More precisely, this research work proposes a generic **modeling approach** that allows application specification and design, regardless of the management platform responsible for application execution, and taking into account the experts involved: domain experts and software developers. The needed separation of domains is achieved by applying model driven engineering. In this way, *domain experts* are in charge of the architectural design of applications which includes all their requirements, whereas *software developers* focus on the design and implementation of the related software solution. Therefore, the modeling approach collects the application requirements that a management platform should assure at runtime, as well as the information needed to generate its skeleton code.

Additionally, this work presents an **Integrated Development Environment** (IDE) that, based on the modeling approach, supports application development cycle. An IDE prototype has been implemented and tested through the specification and design of a demonstrator on the homecare application domain.

ÍNDICE

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN

"Todas las cosas son imposibles, mientras lo parecen."

(Concepción Arenal)

1.1 Motivación

En las últimas décadas se ha producido una auténtica revolución tecnológica, marcada por los numerosos avances científicos producidos en varios ámbitos, entre los que cabe destacar el de la informática, las telecomunicaciones y la microelectrónica. De hecho, avances tales como el despliegue de las comunicaciones inalámbricas, la mejora de la capacidad de procesamiento de los dispositivos, preferentemente de los dispositivos móviles, y el desarrollo de precisos mecanismos de medición han permitido la puesta en marcha de los llamados **sistemas sensibles al contexto** (*context-aware systems*) (Baldauf et al., 2007). Más concretamente, sistemas con aplicaciones que no sólo monitorizan su entorno físico para procesar la información capturada, sino que también deben evolucionar para adaptarse a cambios en él y/o incluso intervenir en su comportamiento, mediante los mecanismos de actuación adecuados. Por lo tanto, proporcionan una respuesta rápida y eficaz ante cambios en su entorno, sin necesidad de intervención humana, lo cual, por otro lado, puede conllevar innumerables beneficios económicos y sociales. En definitiva, las aplicaciones sensibles al contexto presentan tres objetivos principales: (1) **monitorización**, (2) **detección temprana** y (3) **reacción rápida y adecuada**.

Este tipo de aplicaciones se presentan en diferentes ámbitos, como por ejemplo los tres que se tratan en el presente trabajo:

- asistencia domiciliaria: El continuo envejecimiento de la población se está convirtiendo en uno de los mayores retos de los países desarrollados, ya que supone una gran carga social y financiera (United Nations, 2001; World Health Organization, 2011). De hecho, las personas mayores demandan servicios médicos de larga duración que implican grandes gastos en los sistemas públicos de salud, al mismo tiempo que aspiran a mantener su independencia, sin abandonar su hogar. Ante esta problemática, la asistencia domiciliaria se centra en proporcionar *cuidados preventivos* orientados al *reconocimiento del deterioro de la salud*, permitiendo una *atención personalizada*, incluso en caso de emergencia. Para ello se basan en las llamadas casas inteligentes que, integrando inteligencia ambiental

y control automático, permiten proporcionar los cuidados médicos necesarios dentro del hogar, al mismo tiempo que se mejora la seguridad, autonomía y confort de los ancianos (Chan et al., 2008; De Silva et al., 2012).

- sistemas de alerta temprana de desastres medio-ambientales: los desastres medio-ambientales, como por ejemplo incendios e inundaciones, suponen no sólo pérdidas económicas, sino también pérdidas de vidas. Las aplicaciones de alerta temprana se desarrollan para poder prevenir estos desastres y, en caso necesario, actuar rápida y correctamente con el objetivo de minimizar su impacto (Noran, 2014; UNEP, 2012). De hecho, el avance de las comunicaciones inalámbricas ha permitido la *monitorización remota* de áreas de muy diferentes características, desde sistemas de alcantarillado o diques de contención hasta áreas despobladas y hostiles. En base a toda esta información capturada, se pueden *identificar situaciones peligrosas* de forma temprana, proporcionando una *respuesta rápida y apropiada* al nivel de alerta y a la población a la que va dirigida.

- video-vigilancia (multimedia aplicada en vigilancia): la seguridad es uno de los aspectos más críticos en determinados ámbitos como por ejemplo centros penitenciarios, hospitales, bancos, aeropuertos, etc. En estos entornos, la video-vigilancia se plantea como una solución fiable ya que saca partido de las avanzadas técnicas de procesamiento de video actuales (Lim et al., 2014). Este es el caso de la biometría que facilita la identificación de personas en base a sus rasgos faciales, pudiendo emplearse para control de acceso (Connolly et al., 2012; Jain et al., 2006); o el seguimiento de objetos en movimiento que permite optimizar la detección de intrusos (Lim et al., 2014). En definitiva, el video capturado permite la *supervisión* de áreas, y su procesamiento posibilita la *identificación* de situaciones que requieren una *reacción adecuada*, como la apertura de puertas a personal autorizado, o el disparo de alarmas en caso de fuga de presos.

Sin embargo, a pesar de tener finalidades muy diferentes, todas estas aplicaciones comparten características y requisitos similares, tanto funcionales como no-funcionales, entre los que destacan sus demandas de flexibilidad. El avance de las

tecnologías de la comunicación ha permitido la transmisión rápida de información entre lugares alejados físicamente, descentralizando las aplicaciones. Esta **distribución** ha posibilitado la convivencia de dispositivos **heterogéneos** con muy distintas capacidades de recursos (desde sensores, actuadores y dispositivos empotrados hasta equipos con gran capacidad de procesamiento). Las aplicaciones se deben **personalizar** con respecto a las características concretas del usuario y/o entorno. Por ejemplo, el ritmo cardiaco se mide siempre de la misma manera, y sin embargo, la respuesta ante un valor concreto de pulso depende de varios factores como la edad del paciente, la cantidad de ejercicio realizado o si está en reposo. Del mismo modo, la interpretación de un valor de 35ºC para la temperatura ambiental varía si la medida corresponde al interior de un edificio climatizado o si corresponde al desierto del Mojave en California (considerada por los expertos como una de las zonas más calientes del planeta). Finalmente, son aplicaciones que demandan flexibilidad para **evolucionar con los cambios en su contexto** y **disponibilidad** (a pesar de fallos o falta de recursos). A modo de ejemplo, en caso de un alto ritmo cardiaco, además de avisar al personal médico y aumentar la frecuencia de lectura del pulso, se puede incluso considerar necesaria la monitorización del resto de constantes vitales para conocer el estado general del paciente. En un centro penitenciario, la detección de una posible fuga de un preso implica el activar nuevos mecanismos de vigilancia que aumenten el nivel de seguridad del recinto. De forma similar, el cambio de la velocidad y/o dirección del viento puede provocar un cambio en la velocidad y dirección de propagación, e intensidad de un incendio, frente al cual se debe reaccionar.

El disponer de buenas técnicas y metodologías de diseño y desarrollo para este tipo de aplicaciones tan complejas resulta fundamental, sobre todo teniendo en cuenta que se hacen necesarios diferentes expertos que tienen que interactuar. Para una correcta monitorización y detección de situaciones de peligro es necesaria la participación de **expertos del dominio**. Es decir, son los expertos del dominio los que conocen qué se debe monitorizar, cómo detectar cambios relevantes y cómo adaptarse a ellos. Por lo tanto, deben ser los encargados de la *definición de las aplicaciones*. Nótese que esta definición es *independiente de la tecnología* a emplear

para implementarlas. Por ello, son los **desarrolladores de software**, que conocen la tecnología para resolver un problema, los que deben *diseñar y desarrollar la implementación* de las aplicaciones, tomando como punto de partida la especificación realizada por el experto del entorno. Por ejemplo, los profesionales médicos poseen el conocimiento para definir, por cada paciente, cuándo se deben tomar las medidas, cómo procesarlas, cómo reconocer situaciones peligrosas y cómo reaccionar frente a ellas. En el caso de detección de incendios forestales, los bomberos o ingenieros forestales conocen qué medios de detección son adecuados a la región, cómo se propaga un incendio, y cómo hacerle frente. Por último, el personal de seguridad de un aeropuerto es el encargado de definir los mecanismos de vigilancia para la identificación de situaciones peligrosas y los protocolos de actuación si se produce una amenaza. En todos estos casos, los desarrolladores diseñan los módulos necesarios para poder llevar a cabo dichas actividades y los implementan teniendo en cuenta la tecnología subyacente.

Precisamente, el presente trabajo de investigación **estudia** la problemática global de las aplicaciones sensibles al contexto, proponiendo una **metodología y mecanismos** para facilitar su **especificación y diseño**, teniendo en cuenta los diferentes aspectos que intervienen. No sólo se tienen en cuenta sus requisitos funcionales, sino también los no-funcionales que determinan su calidad de servicio (*Quality of Service* – QoS) tales como su distribución geográfica y flexibilidad para adaptarse al contexto; abstrayendo al experto del dominio de los detalles de la tecnología y automatizando, hasta donde sea posible, la **generación del código**. Todo ello apoyado en **herramientas de soporte**.

1.2 Objetivos

Es importante señalar que en la presente memoria se asume que existe una *plataforma de gestión* de la ejecución de las aplicaciones. El diseño e implementación de dicha plataforma de gestión es objeto de otros trabajos en curso, englobados dentro de la misma línea de investigación del Grupo de Control e Integración de Sistemas (GCIS) del Departamento de Ingeniería de Sistemas y Automática de la

Universidad del País Vasco, y con los que el presente trabajo se relaciona. De hecho, los conceptos que se proponen en este trabajo se han validado en casos de estudio que utilizan prototipos de dichas plataformas de gestión.

La visión completa que tienen las plataformas de gestión acerca de las aplicaciones procede, en parte, de su diseño, siendo uno de los resultados de este trabajo el establecimiento de los requisitos a cumplir. Así, el tener en cuenta todos los requisitos de las aplicaciones desde las primeras fases de su diseño y especificación, por un lado facilita su implementación, y por otro, permite establecer los lazos entre su diseño y ejecución.

Por lo tanto, el objetivo general del presente trabajo se centra en la definición de una **metodología**, soportada por **mecanismos** y **herramientas**, que cubra el **ciclo de desarrollo de aplicaciones distribuidas sensibles al contexto**, con objeto de permitir al **experto de dominio** la definición del diseño arquitectónico de dichas aplicaciones, de una forma cercana a su área de conocimiento (abstrayéndolo de los detalles tecnológicos de su implementación), y facilitando, al mismo tiempo, su implementación por parte de los **desarrolladores de software**.

Con el fin de lograr este objetivo principal, se plantean los siguientes objetivos parciales:

- con respecto a las *aplicaciones distribuidas sensibles al contexto*, se debe proporcionar soporte para:
  - ✓ la utilización de plataformas distribuidas y heterogéneas.
  - ✓ la definición de sus requisitos funcionales y no-funcionales; haciendo especial énfasis en sus demandas de flexibilidad tales como disponibilidad, evolución con cambios en el contexto, escalabilidad, personalización, entre otros.
  - ✓ gestionar aplicaciones con estado. En ocasiones, la ejecución de una aplicación depende del resultado de ejecuciones previas. Una correcta

gestión de dicho estado resulta esencial para la recuperación frente a situaciones tales como caídas de nodo.

- ✓ habilitar la monitorización del estado de ejecución de las aplicaciones (interactuando con la plataforma de gestión de su ejecución).

- ❖ en lo que se refiere a los responsables del diseño de las aplicaciones, *expertos de dominio*, se debe facilitar:
  - ✓ abstracción de las tecnologías subyacentes.
  - ✓ herramienta gráfica para el diseño de aplicaciones usando un lenguaje acorde a su área de conocimiento.

- ❖ por su parte, para los *desarrolladores de software* se demanda:
  - ✓ diseño de la solución software (SW) independiente de la plataforma.
  - ✓ herramienta que automatice la generación de código para una plataforma de gestión concreta.

Finalmente, destacar que las diferentes herramientas se deben diseñar con el objetivo de permitir su extensión y personalización para un campo de aplicación y/o plataforma de gestión concretos.

1.3 Estructura

Una vez definidos la motivación y los objetivos del trabajo de investigación, en el segundo capítulo - **Estado del Arte** - se revisan diferentes aproximaciones que tratan de resolver las demandas de este tipo de aplicaciones. En este capítulo se presentan diferentes arquitecturas SW para la implementación de aplicaciones distribuidas y se analizan trabajos relacionados que presentan una propuesta de diseño o gestión de los requisitos de QoS de las aplicaciones de interés: adaptación al contexto, gestión de calidad de servicio específica de aplicación, disponibilidad y seguridad. Al mismo

tiempo se estudia si las plataformas de gestión analizadas disponen de soporte al desarrollo.

El tercer capítulo – **Aproximación de modelado: CADAMM** - está dedicado a la primera contribución del presente trabajo: la aproximación de modelado CADAMM (*Context-Aware Distributed Applications Meta-Model*) que ofrece los mecanismos necesarios para la especificación y diseño de aplicaciones distribuidas sensibles al contexto. Inicialmente, se analizan aplicaciones de este tipo pertenecientes a los tres ámbitos de aplicación previamente comentados (asistencia domiciliaria, sistemas de alerta temprana de desastres medio-ambientales y video-vigilancia), con el objetivo de identificar situaciones relevantes (reales y que suponen un verdadero reto) de las que extraer *requisitos comunes* a este tipo de aplicaciones. A continuación, se describen los diferentes *elementos de modelado* propuestos para cumplir con dichos requisitos, su *caracterización* y las *relaciones* entre dichos elementos, teniendo en cuenta los *puntos de vista* de los diferentes expertos que participan: expertos de dominio que llevan a cabo el diseño arquitectónico de las aplicaciones y desarrolladores de software que diseñan e implementan la solución SW correspondiente a dicho diseño arquitectónico. Resulta importante destacar, que a pesar de ser el resultado del análisis de campos concretos, tanto los requisitos como la aproximación de modelado son aplicables a otros campos. De hecho, la solución propuesta tiene en cuenta la extensión de características del dominio proporcionando mecanismos para definir QoS específica.

El cuarto capítulo – **Entorno de Desarrollo Integrado: CADAMToolSuite** – está dirigido al diseño de las herramientas de soporte al desarrollo correspondientes a la aproximación de modelado CADAMM, que se pueden agrupar en un entorno de desarrollo integrado (*Integrated Development Environment*, IDE). Dicho entorno sigue un diseño modular que independiza unos módulos de otros de manera que únicamente interactúan a través del modelo de aplicaciones generado, facilitando la extensión y personalización de CADAMToolSuite. Más concretamente, el IDE dispone de un editor que guía al experto de dominio durante el diseño arquitectónico de las aplicaciones, asegurando la construcción de modelos correctos. También permite que

el desarrollador de software describa el diseño detallado e implementación correspondiente a la especificación del experto de dominio. Finalmente, dispone de un generador automático del esqueleto del código de las aplicaciones y un módulo para la gestión de la ejecución de dichas aplicaciones (arranque/parada). Estos dos módulos se deben personalizar para una plataforma de gestión concreta ya que por un lado es la propia plataforma de gestión la que establece la estructura del código generado, y por otro lado, es necesario interactuar con ella para el registro de la información de diseño así como para el arranque y parada de las aplicaciones. De hecho, se han realizado pruebas con dos prototipos de plataforma de gestión diferentes: uno basado en componentes y otro basado en agentes.

Para probar la validez de la aproximación propuesta, el quinto capítulo – **Caso de Estudio: Asistencia Domiciliaria** – presenta un prototipo del entorno CADAMToolSuite para el campo de la asistencia domiciliaria, utilizando una plataforma de gestión basada en agentes. Más concretamente, se presenta el diseño y desarrollo de un demostrador centrado en una residencia de ancianos, cuyas aplicaciones contemplan los requisitos identificados.

Finalmente, en el capítulo final – **Conclusiones y Líneas Futuras** – se agrupan las conclusiones y aportaciones del trabajo, así como posibles futuros trabajos relacionados.

2 ESTADO DEL ARTE

"El poder de cuestionar es la base de todo progreso humano."

(Indira Gandhi)

2.1 Introducción

En este capítulo se analizan trabajos relacionados que abordan los requisitos demandados por las aplicaciones de interés, ya identificados en el Capítulo 1. El objetivo del presente capítulo es doble. Por un lado, determinar qué es necesario capturar en fase de diseño para que dichos requisitos se cumplan en ejecución. Y por otro, identificar qué aspectos relativos a dichos requisitos no se cubren en la literatura.

Desde el punto de vista de la implementación se han propuesto **arquitecturas software** que tienen en común el considerar una aplicación como un conjunto de módulos (elementos computacionales) que pueden ejecutarse en diferentes nodos, y deben interactuar para lograr el objetivo de la aplicación. El conjunto de los módulos de una aplicación y sus interconexiones definen su *configuración*. Así, cualquier implementación de una de estas arquitecturas SW permite la ***distribución*** y ***personalización*** de la funcionalidad de las aplicaciones, requisitos ambos de las aplicaciones de interés. De hecho, en tiempo de ejecución, se emplean **plataformas de gestión** de la ejecución de las aplicaciones que implementan una arquitectura SW concreta, y que proporcionan mecanismos para, como mínimo, controlar el ciclo de vida de sus módulos, facilitar su despliegue en los nodos de la infraestructura y posibilitar la comunicación entre ellos.

Además, se han desarrollado plataformas de gestión que también incorporan cierta gestión de la QoS de las aplicaciones, para lo que se basan en mecanismos de reconfiguración dinámica que permiten hacer evolucionar una aplicación de una configuración a otra, sin por ello detener su ejecución (Almeida et al., 2004; Li, 2011; Wegdam et al., 2003). En este sentido, algunas plataformas de gestión se enfocan en hacer ***evolucionar*** a las aplicaciones ***con cambios en su contexto***, estando, habitualmente, muy ligadas a las particularidades de un contexto concreto. Otras permiten gestionar la ***QoS específica de un campo o tipo de aplicación determinado***, mediante el control de los recursos demandados por las aplicaciones y el de los disponibles en el sistema, en ocasiones incluso para aquéllas que soportan

cierta degradación de su QoS. En otros casos, las plataformas de gestión se centran en *asegurar la* ***disponibilidad*** de las aplicaciones o recuperar su ejecución en caso de fallo de nodo, generalmente mediante la gestión de la redundancia (a nivel funcional o a nivel de nodo).

En resumen, las plataformas de gestión velan por el cumplimiento de los requisitos de las aplicaciones, gracias a que disponen de una visión y control completo tanto de las aplicaciones como del sistema. Estos requisitos los debe imponer el experto de dominio en fase de diseño, en base a los cuales el desarrollador de SW implementa las aplicaciones. Sin embargo, a pesar de que existen entornos de soporte que facilitan el diseño y/o automatizan su desarrollo, no proporcionan una clara ***separación de dominios***, ya que el diseño o está muy enfocado a un experto de dominio con altos conocimientos de la tecnología, o está directamente orientado al desarrollador de SW.

Con objeto de analizar en qué grado las diferentes arquitecturas y plataformas de gestión existentes soportan la especificación, diseño y gestión de la ejecución de las aplicaciones de interés, este capítulo se divide en tres bloques fundamentales. En primer lugar se presentan diferentes aproximaciones para la composición y gestión de aplicaciones distribuidas, sobre las cuales se apoyan las plataformas de gestión. En el segundo bloque se analizan trabajos de otros autores relacionados con el diseño y gestión de requisitos no estrictamente funcionales, principalmente de aquéllos demandados por las aplicaciones de interés. El último bloque se dedica a entornos que dan soporte al diseño y/o desarrollo de las aplicaciones.

2.2 Composición y Gestión de Aplicaciones Distribuidas

Como se ha comentado en el Capítulo 1, el avance de las tecnologías ha permitido la descentralización de las aplicaciones, de manera que la coordinación de las aplicaciones distribuidas ha recibido gran atención por parte de la comunidad científica. Así, se han propuesto arquitecturas SW que facilitan la **distribución**,

escalabilidad y **personalización** de la funcionalidad de las aplicaciones. En todas ellas se propone la definición (1) de los bloques funcionales individuales y (2) de cómo se componen para constituir una aplicación.

En la **definición de los bloques** se persigue la *reutilización* de software, siendo necesaria una descripción detallada de la *interfaz funcional* del bloque, y estableciendo una clara separación entre la funcionalidad proporcionada y su implementación. Con respecto a su **composición**, es necesario definir las *relaciones entre los bloques* que interactúan directamente intercambiando datos, así como la arquitectura de sistema a seguir.

En los siguientes sub-apartados se analizan estos conceptos en el marco de las arquitecturas SW más usadas: basada en componentes, orientada a servicios y basada en agentes.

2.2.1 Sistemas Basados en Componentes

La ingeniería de software basada en componentes (*Component-Based Software Engineering*, CBSE) propone la construcción de sistemas complejos mediante la **composición** de bloques simples (**componentes**), previamente desarrollados independientemente de la aplicación en la se vayan a usar (Szyperski, 1998). De hecho, la separación entre interfaz e implementación propuesta por CBSE (Bachmann et al., 2000) permite que un componente se conciba como una *caja negra* que encapsula servicios, de manera que para hacer uso de él no sea necesario conocer sus detalles internos sino que baste con la caracterización de su interfaz. Por lo tanto, se identifican dos procesos principales: (1) el desarrollo de *componentes* software *reutilizables* y *distribuibles*, y (2) el desarrollo de las aplicaciones como composición de dichos componentes. Se trata de dos procesos independientes, llevados a cabo por distintos agentes y en tiempos diferentes, pero que se deben complementar y coordinar. Sin embargo, no existe un consenso en cuanto a la definición de componente, tal y como se recoge en Lau & Wang (2007), si bien se puede destacar la propuesta por Heineman & Councill (2001) en la que se liga a un **modelo de**

componentes que proporciona *reglas* tanto para la construcción de componentes individuales como para su composición. Por lo tanto, un modelo define los tipos de componentes que existen, el modo en que éstos se especifican, y los patrones de interacción entre ellos y con el entorno de ejecución (Crnkovic et al., 2011).

Por ejemplo, las especificaciones propuestas por la organización para el avance de estándares para la información estructurada (*Organization for the Advancement of Structured Information Standards*, OASIS) describen el modelo de componentes *Service Component Architecture* (SCA) (OASIS, 2007) que separa la lógica de los componentes de la lógica de comunicación. Más concretamente, en base a ficheros de configuración, se define un componente como un elemento software que expone servicios a los demás, al mismo tiempo que precisa servicios proporcionados por otros (llamados referencias), estableciéndose las comunicaciones entre componentes mediante conexiones (*wires*) entre servicios y referencias. Nótese que SCA permite configuraciones estáticas por lo que no es posible que la estructura de la aplicación cambie en tiempo de ejecución. Sin embargo, existen algunos modelos de componentes que sí lo contemplan como es el caso de FRACTAL (Blair et al., 2009; Bruneton et al., 2006) y SOFA2.0 (Bures et al., 2006). Ambos distinguen una interfaz funcional y otra de control para los componentes que el desarrollador debe definir, dotándoles de capacidades de control de su ejecución y estructura interna. En cualquier caso, también se pueden proponer modelos de componentes que contemplen aspectos no-funcionales de los componentes, algunos de los cuales se analizan en detalle dentro del apartado 2.3, como por ejemplo el modelo Pecos (Wuyts & Ducasse, 2001) que contempla la QoS de sistemas empotrados, el propuesto para el framework ACCADA (Gui et al., 2011) que considera sus demandas de recursos. Otros trabajos se centran en un determinado campo de aplicación como el propuesto en López et al. (2013) que aborda la definición del modelo de tiempo real de una aplicación como composición de los modelos de tiempo real de los componentes que la conforman.

En tiempo de ejecución, las **plataformas de gestión** de aplicaciones, implementaciones de modelos de componentes, deben ofrecer mínimamente

servicios de instanciación, ensamblado y gestión del ciclo de vida, tal y como ofrece Tuscany (Laws et al., 2011), la implementación de referencia para el estándar SCA, o FraSCAti (Seinturier et al., 2009), otra implementación de SCA que además soporta reconfiguración dinámica, extendiendo dicho modelo de componentes con una arquitectura similar a Fractal. Es más, existen otras plataformas de gestión, analizadas en detalle en el apartado 2.3, que proporcionan *servicios de valor añadido* como la gestión de la adaptación al contexto (Gui et al., 2011; Hofmeister, 1998; Khan et al., 2008; Léger et al., 2010; Wegdam et al., 2003), la gestión dinámica de su QoS (Gui et al., 2011; Hallsteinsen et al., 2012; Kon et al., 2005; Mitchell et al., 1999; Noguero et al., 2013; Tamura et al., 2014) o la disponibilidad de las aplicaciones (Cervantes & Hall, 2004; Hallsteinsen et al., 2012; Noguero et al., 2013).

2.2.2 Arquitecturas Basadas en Servicios

El paradigma de la computación orientada a servicios (*Service Oriented Computing*, SOC) promueve la compartición y reutilización de unidades software llamadas **servicios** para dar soporte al desarrollo de aplicaciones distribuidas. Más concretamente, en SOC los servicios son unidades computacionales *independientes de la plataforma* que pueden ser publicados en *repositorios* por parte de sus proveedores, de manera que los consumidores pueden descubrirlos, seleccionarlos, hacer uso de ellos o componerlos para proporcionar nuevos servicios (Papazoglou et al., 2007). Al igual que ocurre con los componentes, en las arquitecturas orientadas a servicio (*Service Oriented Architecture*, SOA) los servicios también se consideran *cajas negras*, ya que sus consumidores no deben conocer los detalles de su implementación (Erl, 2005). Únicamente es necesaria una *descripción* que contenga aquella información que permita decidir si se trata del servicio adecuado, así como información que especifique cómo poder interactuar con él (interfaz, comportamiento y localización). Sin embargo, comparado con el rígido ensamblado propuesto por CBSE, una SOA se presenta como una propuesta mucho más flexible, ya que la selección del proveedor se lleva a cabo justo cuando se precisa un servicio (Estublier & Vega, 2012; Fiadeiro & Lopes, 2013).

Por otro lado, en la **composición** de aplicaciones en base a un *conjunto de servicios débilmente acoplados* se distinguen dos aproximaciones principales (Dijkman & Dumas, 2004; Fiadeiro & Lopes, 2013): la orquestación y la coreografía. En el primer caso un orquestador interactúa con todos los servicios estableciendo el flujo de ejecución, mientras que en el segundo caso se describen los mensajes intercambiados entre varios servicios para lograr un determinado objetivo.

En este contexto, en la literatura se pueden encontrar propuestas para la especificación de servicios, y para el diseño y/o análisis de composiciones de servicios. Con respecto a la **descripción de servicios**, el lenguaje de descripción de servicios web, la principal tecnología de implementación de SOA, (*Web Services Description Language*, WSDL) (W3C, 2001) proporciona una gramática *eXtensible Markup Language* (XML) (W3C, 2006) para su definición como un conjunto de puntos de comunicación (puertos) capaces de realizar operaciones en base al intercambio de mensajes que contienen la información necesaria para ello. Es más, la descripción de los puertos y de los mensajes es independiente del protocolo de red o formato de datos empleado, estableciéndose dicha conexión en base a la definición de los llamados *bindings*. Por su parte, el lenguaje de modelado SOAML (OMG, 2012) para la especificación y diseño de servicios considera una SOA como una red de participantes que proporcionan y consumen servicios para lograr un objetivo. Por lo tanto, SOAML permite definir los diferentes participantes (consumidores y proveedores), las funcionalidades ofrecidas por un servicio, así como los protocolos y la información intercambiada entre proveedores y consumidores.

En cuanto al **diseño y análisis de las composiciones**, en Skogan et al. (2004) se propone un proceso para definir nuevos servicios web mediante la composición de otros previamente definidos y almacenados en repositorio haciendo uso del lenguaje de modelado *Unified Modeling Language* (UML). Inicialmente se debe caracterizar tanto la interfaz del nuevo servicio web como su composición en base a las operaciones que lleva a cabo, de forma que es posible buscar en un repositorio los servicios web adecuados a dichas operaciones. Una vez que la composición ha sido refinada, se genera tanto la descripción WSDL del nuevo servicio web como su código.

Los trabajos presentados en Barkaoui et al. (2010) y Dumez et al. (2013) van un paso más allá y permiten la verificación de la composición de servicios web. Los primeros emplean métodos estructurales basados en redes de Petri, para lo cual necesitan conocer no sólo la interfaz sino también el comportamiento de los componentes. Los segundos proponen definir la composición de servicios mediante diagramas de actividad UML, lo cual permite la generación del código ejecutable de los servicios web. Para poder hacer uso de herramientas de análisis existentes y de probada validez, plantean la transformación del modelo UML a lenguajes formales de definición de composiciones. Por último, la aproximación propuesta en Foster et al. (2011) también emplea métodos formales para el análisis de la composición deg servicios, pero de SOAs dinámicamente reconfigurables. Se introduce el concepto de modo para, en fase de diseño, identificar el conjunto de servicios que deben interactuar en un determinado instante de ejecución, de manera que se entiende la reconfiguración dinámica como un cambio de modo.

En este contexto es importante destacar que la *QoS* de los servicios resulta un factor clave para la composición de aplicaciones, ya que permite seleccionar el servicio más adecuado entre varios que ofrecen la misma funcionalidad. Además permite que en tiempo de ejecución se pueda gestionar dicha QoS de forma dinámica. Es por ello que, al igual que en las aplicaciones basadas en componentes, se han propuesto aproximaciones que capturan la QoS de los servicios, y que se analizan en el apartado 2.3 como es el caso del proyecto Quadrantis (Tran et al., 2009) para aplicaciones multimedia basadas en servicios web. Sin embargo, la mayoría de los trabajos descritos en este apartado está enfocada al caso particular de los servicios web o considera un paradigma de comunicación basado en el intercambio de mensajes entre servicios.

También es necesario destacar que existen otras **plataformas de gestión** de aplicaciones que implementan este modelo arquitectónico, como el trabajo realizado dentro del proyecto iLAND (*mIddLewAre for deterministic dynamically reconfigurable NetworkeD embedded systems*) (García-Valls et al., 2013b) cuyo objetivo es proporcionar mecanismos para la composición y reconfiguración funcional de

aplicaciones distribuidas, de forma dinámica y determinista. Con este objetivo, se propone un modelo de servicio y de aplicaciones (García-Valls, et al., 2013a) así como un middleware que gestiona su ejecución y proporciona reconfiguración acotada en el tiempo (García-Valls & Basanta-Val, 2013). La plataforma MOSES (Cardellini et al., 2012) proporciona gestión dinámica de la QoS al seleccionar la implementación más adecuada para un servicio, teniendo en cuenta que cada implementación presenta diferente QoS y coste computacional. Estos trabajos se analizan en detalle en el apartado 2.3.

2.2.3 Sistemas Multiagente

Aunque no existe una definición de **agente** única, sus principales características se identifican en Wooldridge & Jennings (2009) donde se indica que son: *autónomos*, pudiendo tomar decisiones sin intervención humana directa; *proactivos*, ya que su comportamiento está dirigido por objetivos; *reactivos*, ya que pueden responder a cambios en su entorno; *sociales*, teniendo en cuenta que interactúan entre ellos; y móviles, ya que pueden "viajar" de un nodo a otro a través de la red. Así, un **sistema multiagente** (*Multi-Agent System*, MAS) consiste en dos o más agentes que cooperan para resolver un problema complejo o que compiten para lograr objetivos individuales y/o colectivos, todo ello intercambiando mensajes. Teniendo en cuenta los principales aspectos que caracterizan a los agentes, se considera que los MAS son adecuados para el desarrollo de sistemas distribuidos y adaptables (Weiss, 1999).

Con respecto al **diseño de los MAS** y tal y como se recoge en Isern et al. (2011), en la literatura se pueden encontrar aproximaciones centradas en el diseño de los agentes, aunque actualmente son más comunes las aproximaciones orientadas a las organizaciones de agentes. De hecho, a pesar de que los agentes se consideran entidades autónomas que actúan de acuerdo a determinados objetivos, lo cierto es que también son miembros de una sociedad, por lo que intercambian información con otros agentes, con los que mantienen una relación con un determinado nivel organizativo (Horling & Lesser, 2004). Es por ello que se han propuesto metodologías para el desarrollo de MAS basadas en la proposición de *estructuras organizativas*

(Horling & Lesser, 2004) que determinan las reglas que controlan el flujo de datos entre agentes, la coordinación entre ellos, o la asignación de recursos. Desafortunadamente, no se ha encontrado ninguna metodología que pueda considerarse estándar, si bien todas ellas manejan conceptos similares (Isern et al., 2011): capacidades ofrecidas por un agente (su *rol*), los *grupos* a los que pueden pertenecer, la *interacción* entre agentes así como la *colaboración* y *dependencia* entre los posibles grupos.

En cuanto al **soporte al desarrollo**, es destacable el esfuerzo realizado por la fundación *Foundation for Intelligent Physical Agents* (FIPA) en la estandarización de aspectos relacionados con los agentes, proponiendo un conjunto de especificaciones para la interacción y comunicación entre ellos, así como una arquitectura de referencia (Foundation for Intelligent Physical Agents, 2002).

En tiempo de ejecución se dispone de **plataformas de gestión** que pueden ir desde las que ofrecen funcionalidades básicas (gestión de del ciclo de vida, registro y búsqueda de agentes, gestión de la comunicación distribuida), hasta otras más complejas que aportan servicios de valor añadido. En el primer grupo se encuentra *Java Agent DEvelopment Framework* (JADE) (Bellifemine et al., 2008), el framework de desarrollo de agentes más conocido y usado, que implementa el estándar FIPA. En el segundo grupo se pueden destacar los siguientes trabajos que se analizan en detalle en el apartado 2.3: la plataforma THOMAS (Bajo et al., 2010) que aporta gestión dinámica de organizaciones virtuales de agentes; y el trabajo propuesto en (García-Magariño & Gutiérrez, 2013) que se basa en JADE para proporcionar una gestión óptima de la tolerancia a fallos en situaciones de desastres naturales.

2.3 Gestión de Calidad de Servicio (QoS)

Este apartado se centra en aquellos requisitos que no son estrictamente funcionales, y en particular en los demandados por las aplicaciones de interés, cuyo análisis, presentado en el capítulo 1, ha permitido destacar los siguientes:

- Adaptabilidad: necesidad de evolucionar con cambios en el contexto.

- QoS específica de aplicación: características que dan idea de la calidad de una aplicación y que pueden variar de una aplicación a otra.
- Disponibilidad: recuperación frente a fallo de nodo o componente.
- Seguridad: integridad y privacidad de los datos gestionados por las aplicaciones, y seguridad en el acceso a dichos datos y recursos del sistema.

Desafortunadamente, no existe un consenso en cuanto a qué se considera **requisito no-funcional** (*Non-Functional Requierment*, NFR), ni por lo tanto en cuanto a cómo debe definirse y capturarse. Incluso en ocasiones resulta complicado establecer una frontera entre ambos tipos de requisitos (Bajpai & Gorthi, 2012). Es más, si se repasan los términos recogidos en el glosario (IEEE, 1990), se observa que sí se propone una definición de requisito funcional (*Un requisito que especifica la función que un sistema o un componente del sistema debe ser capaz de realizar* (IEEE, 1990, p.34)), pero no para los NFR. En los trabajos de Glinz (2007) y Chung & do Prado Leite (2009) se recogen algunas de las definiciones de NFR que se pueden encontrar en la literatura. Son definiciones tan variadas como por ejemplo: "*Los atributos generales requeridos de un sistema, incluyendo su portabilidad, fiabilidad, eficiencia, ingeniería humana, capacidad de realizar ensayos, comprensibilidad y posibilidad de modificación*" ((Davis, 1993) citado en Glinz (2007, p.22)) o "*Los requisitos no-funcionales constituyen las justificaciones de las decisiones de diseño y limitan la forma en que la funcionalidad requerida se puede realizar*" ((Landes & Studer, 1995) citado en Chung & do Prado Leite (2009, p.366)). En general, los NFRs comprenden el conjunto de características que imponen un comportamiento a las aplicaciones, dando idea de su calidad. Es por ello que habitualmente se les conoce como **calidad de servicio**.

Como consecuencia, también se pueden encontrar diferentes **clasificaciones**. Por un lado, existen propuestas institucionales y estándares, como es el caso de la especificación QFTP (OMG, 2008) y el estándar ISO/IEC 25000 SQuaRE (ISO/IEC, 2014), respectivamente. Por otro lado, se pueden encontrar clasificaciones correspondientes a trabajos de investigación, entre las que se pueden destacar las presentadas en Becker (2008), Chalmers & Sloman (1999), Jureta et al. (2006),

Mairiza et al. (2010), Nehmer et al. (2006), Roman (1985) y Wehrmeister et al. (2014). Algunas de estas clasificaciones son *genéricas*, ya que identifican requisitos comunes a cualquier dominio, mientras que otras son *específicas de un ámbito o tipo de aplicación concreto*.

En el grupo de las genéricas se encuentra el trabajo de Jureta et al. (2006) en el que se presenta una clasificación muy general que distingue entre requisitos no-funcionales críticos (*nonfunctional hardgoals*) y no críticos (*nonfunctional softgoals*), en cuanto a si la descripción de cómo se debe proporcionar un servicio se realiza mediante criterios objetivos o declaraciones imprecisas, respectivamente. La serie de estándares SQuaRE (ISO/IEC, 2014) está enfocada a la calidad del software, y distingue seis características de calidad, una funcional y las siguientes cinco no-funcionales: fiabilidad (tolerancia a fallos, disponibilidad), usabilidad, eficiencia/rendimiento (requisitos temporales), mantenimiento y portabilidad. Por su parte, la especificación QFTP (OMG, 2008), con el objetivo de que los modelos de QoS sean reutilizables en diferentes dominios, identifica las categorías de QoS mostradas en la Figura 2-1.

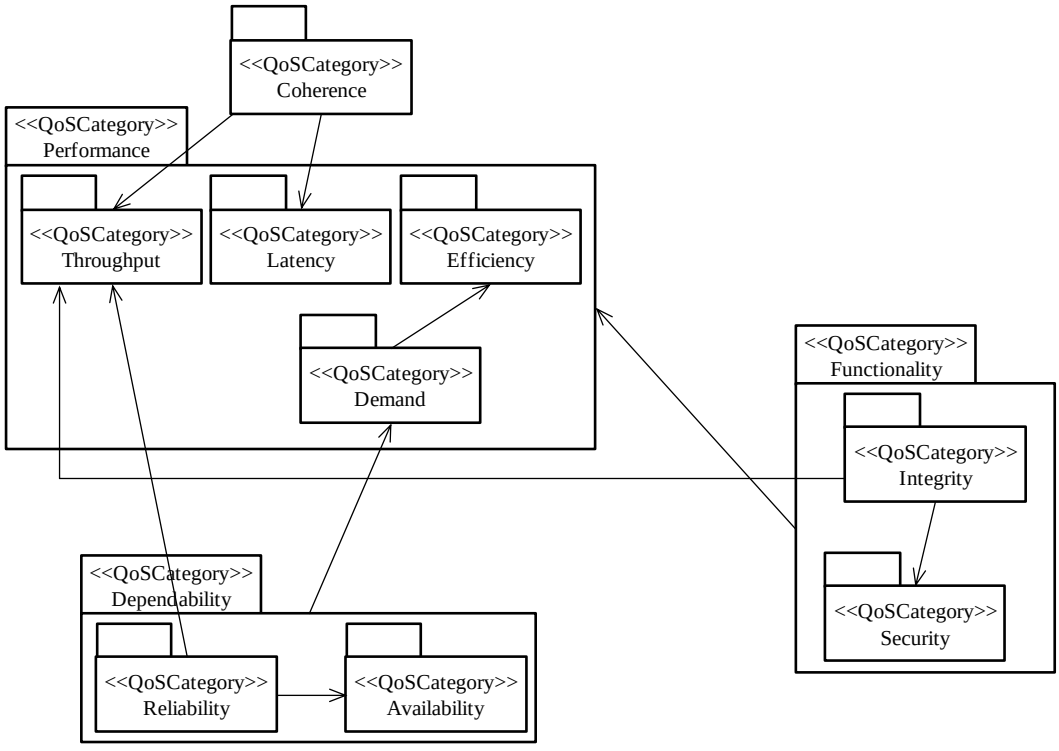


Figura 2-1: Categorías generales de la especificación QFTP (OMG, 2008)

Por su parte, Roman (1985) entiende los NFR como restricciones que clasifica en los siguientes grupos: de interfaz (cómo interactúan los componentes y el sistema), de rendimiento (requisitos temporales, seguridad, disponibilidad, etc.), operacionales (p. ej. accesibilidad), del ciclo de vida (mantenimiento, flexibilidad, portabilidad, etc.), económicas (costes a corto y largo plazo) y políticas (cuestiones legales). En Mairiza et al. (2010) se analizan trabajos de la literatura con el objetivo de identificar los diferentes atributos (tipos) empleados para definir NFR (ver Figura 2-2), y se agrupan en categorías, señalando las más usadas: eficiencia, fiabilidad, usabilidad, seguridad y mantenimiento.

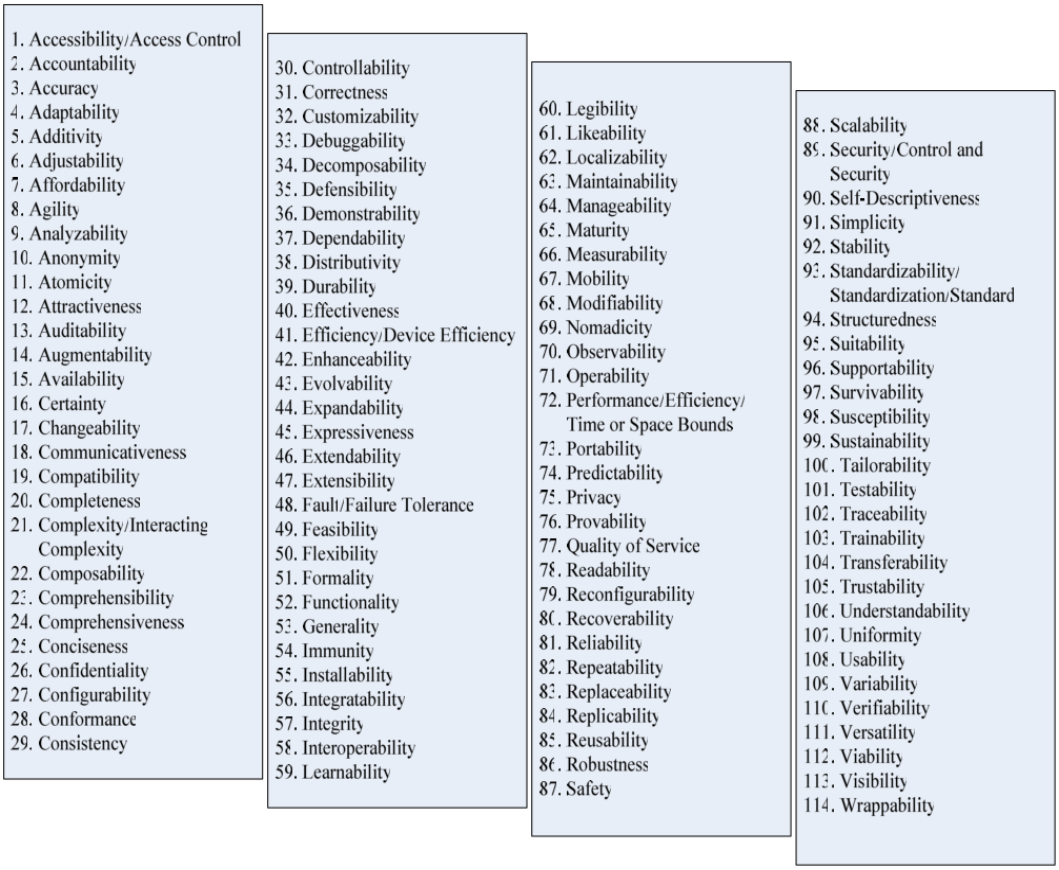


Figura 2-2: Lista de tipos de requisitos no-funcionales (Mairiza et al., 2010, p.313)

En el segundo grupo, clasificaciones específicas de campo de aplicación, se engloban los trabajos de Nehmer et al. (2006) y Becker (2008) para el campo de ayuda a la vida cotidiana asistida por el entorno (*Ambient Assisted Living*, AAL), para el que, además de muchos de los requisitos genéricos ya presentados, se resaltan los de extensibilidad, eficiencia de recursos y adaptabilidad. Chalmers & Sloman (1999) se

centran en el análisis de aplicaciones multimedia para las que resulta importante la fiabilidad y el respeto de plazos. Además, en este tipo de aplicaciones aparece el concepto de **calidad percibida** que puede ser representada por atributos tales como la frecuencia del video, el detalle de la imagen, la frecuencia de muestreo del audio y la precisión del color. Por último, en Wehrmeister et al. (2014) se presenta la clasificación mostrada en la Figura 2-3 para sistemas empotrados de tiempo real, siendo los requisitos temporales clave en este tipo de aplicaciones.

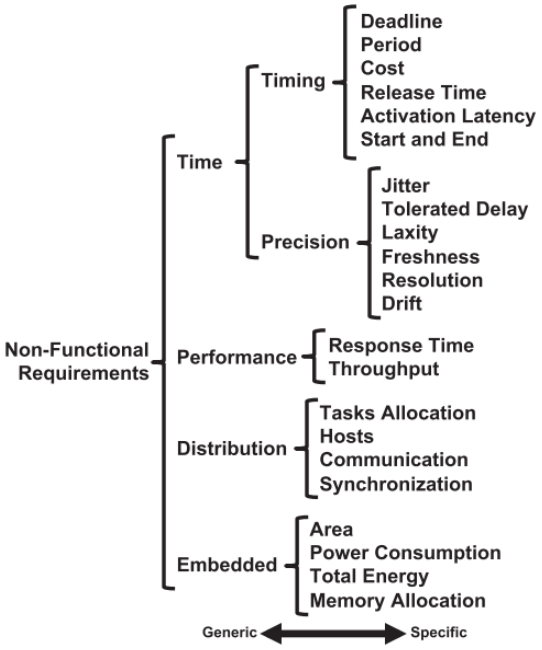


Figura 2-3: Requisitos no-funcionales para sistemas empotrados de tiempo real (Wehrmeister et al., 2014, p.848)

Como se puede deducir de esta selección de trabajos, independientemente de su clasificación, en torno a la QoS se engloban características muy diferentes, algunas de las cuales son genéricas a todos los ámbitos (disponibilidad, requisitos temporales, seguridad, fiabilidad, etc.) mientras que otras son específicas a determinadas aplicaciones (calidad del video en aplicaciones multimedia, adaptabilidad en aplicaciones sensibles al contexto como en los sistemas AAL, etc.). Además, algunas de estas características hacen referencia a *todo el sistema*, como la necesidad de eficiencia energética o la disponibilidad del sistema, mientras que otras únicamente se refieren a *determinadas partes del sistema* (p. ej. frecuencia con la que se debe realizar una medida y la privacidad de los datos procesados).

Por otro lado, resulta importante señalar que las aplicaciones distribuidas sensibles al contexto se engloban dentro de los llamados **sistemas auto-adaptables** (*self-adaptive systems*). En la literatura se pueden encontrar diferentes definiciones para este tipo de sistemas y en todas ellas se hace referencia a cambios en tiempo de ejecución que pueden afectar a los requisitos bien funcionales bien no-funcionales de las aplicaciones (Krupitzer et al., 2015; Macías-Escrivá et al., 2013). En el presente trabajo se adopta la definición propuesta por Krupitzer et al. (2015) que entiende como sistema auto-adaptable aquel capaz de *modificarse a sí mismo, de forma automática, en respuesta a cambios en su entorno de operación*. Por lo tanto, estos sistemas requieren tener conciencia de sí mismos (*self-awareness*) así como de su entorno (*context-awareness*). Es decir, deben ser conscientes de su estado mediante la monitorización de sus recursos y comportamiento, así como conscientes de su entorno mediante la monitorización de su contexto. Así, un diseño, desarrollo y gestión correctos del proceso de adaptación en este tipo de sistemas permite considerar la mayor parte de los requisitos de interés en el presente trabajo. En particular, los de adaptabilidad, gestión de la QoS específica de aplicación y disponibilidad.

Tal y como se recoge en Salehie & Tahvildari (2009), a la hora de definir un proceso de adaptación se deben tener en cuenta seis cuestiones, que se explican a continuación:

- ***Por qué (Why)***

  Cuál es el *objetivo perseguido* por un proceso de adaptación. Más concretamente, la adaptación se produce como *reacción a un cambio*, que puede tener diferentes orígenes: cambios en el contexto, cambios en los recursos del sistema o cambios en las preferencias de usuarios. Los cambios en las preferencias de usuarios quedan fuera del ámbito del presente trabajo de investigación. La adaptación a cambios en el contexto (*adaptabilidad*) es objeto del apartado 2.3.1, mientras que la adaptación para soportar cambios en los recursos disponibles en el sistema (*gestión dinámica de la QoS o QoS flexible*) se trata en el apartado 2.3.2, excepto el caso particular de los problemas de *disponibilidad* que se abordan en el apartado 2.3.3.

- ***Quién (Who)***

Hace referencia al grado de autonomía o intervención humana. Obviamente, siendo uno de los objetivos que las aplicaciones de interés sean capaces de adaptarse para cumplir sus requisitos, se espera que sea una adaptación *automática*, *sin intervención humana*.

- ***Cómo (How)***

Los procesos de adaptación son complejos y requieren varias tareas: monitorización, análisis, planificación y ejecución. Aunque existen aproximaciones que incorporan el proceso de adaptación dentro del código de la aplicación, la mayoría de los trabajos abogan por *separar la lógica de la aplicación de la lógica de adaptación* (Kon et al., 2005; Macías-Escrivá et al., 2013; Wegdam et al., 2003). De esta manera, un gestor externo a la aplicación, una *plataforma de gestión de ejecución*, se encarga de controlar el proceso de adaptación. El principal mecanismo de adaptación es la **reconfiguración dinámica** (Almeida et al., 2004; Li, 2011; Wegdam et al., 2003). En general, dentro de un proceso de reconfiguración dinámica se distinguen cuatro posibles operaciones: crear, eliminar, reemplazar (cambiar la versión de una entidad) y migrar (hacer que una entidad o una nueva versión de ella se ejecute en otro nodo) (Wegdam et al., 2003).

- ***Cuándo (When)***

Se refiere a cuándo existe la necesidad de llevar a cabo una adaptación y a cuándo es posible realizar las operaciones de adaptación necesarias. A este respecto, el presente trabajo se centra en aproximaciones *reactivas*, en las que primero se detecta un suceso relevante y después se reacciona (Salehie & Tahvildari, 2009). Por lo tanto, para saber si una adaptación es necesaria se debe monitorizar el entorno y así reaccionar frente a cambios relevantes en él (Augusto & Nugent, 2004; Benghazi et al., 2012; Botia et al., 2012; Rabbi et al., 2014; Søberg et al., 2010), y/o monitorizar el estado de los recursos del sistema para una correcta gestión de la QoS o recuperarse de situaciones de fallo (Bajo et al., 2010; Cervantes & Hall, 2004; García-Valls, Rodríguez-López, et al., 2013; Gui et al.,

2011; Tamura et al., 2014; Tran et al., 2009). En cuanto a cuándo ejecutar las operaciones necesarias, queda en manos de la plataforma de gestión el planificar y ejecutar el proceso de adaptación. Además, es imprescindible que, en cualquiera de estas situaciones, el proceso de adaptación mantenga la consistencia del sistema (Kramer & Magee, 1990; Léger et al., 2010; Vandewoude et al., 2007; Wegdam et al., 2003). Todas estas situaciones y trabajos se analizan a lo largo de los siguientes sub-apartados.

- ***Dónde (Where)***

  Qué entidades se ven afectadas por la adaptación: aplicaciones, recursos físicos, componentes... Habitualmente, se actúa sobre componentes (Hofmeister, 1998; Khan et al., 2008), si bien existen trabajos que intervienen a nivel de aplicación (Bajo et al., 2010; García-Valls, Rodríguez-López, et al., 2013), principalmente para acciones de crear o eliminar. En cuanto a los recursos físicos, en la mayoría de las ocasiones se consideran los receptores de las instancias en ejecución (Kon et al., 2005). Por último, el entorno también puede modificarse mediante actuadores (Augusto & Nugent, 2004; Pérez et al., 2009), pero ese caso se puede comprender dentro de la funcionalidad de la propia aplicación.

- ***Qué (What)***

  Qué acciones hay que llevar a cabo sobre dichas entidades. En este contexto, las ya citadas operaciones de crear, eliminar, reemplazar y migrar (Wegdam et al., 2003) pueden dar lugar a cambios en la estructura de una aplicación (Gui et al., 2011; Léger et al., 2010) o incluso cambios en la arquitectura del sistema (Bajo et al., 2010; García-Valls et al., 2013b). Además, también es posible modificar los parámetros de los elementos, sobre todo los relacionados con su QoS (Foster et al., 2011).

A pesar de toda la diversidad que rodea a la QoS, en lo que sí hay consenso en la literatura es en que no es sólo un aspecto a tener en cuenta en tiempo de ejecución, sino que también debe ser considerada desde las primeras fases de diseño, tal y como se comenta en (Bajpai & Gorthi, 2012; Chung & do Prado Leite, 2009; de Souza Neto,

2012; Li & Guo, 2015). Por ello, a lo largo de los siguientes sub-apartados se analizan trabajos relacionados con el diseño y gestión de los requisitos de QoS identificados: adaptación al contexto, QoS específica de aplicación, disponibilidad y seguridad. Mención especial recibe la transferencia del estado que garantiza la consistencia de las aplicaciones tras un proceso de reconfiguración o recuperación, independientemente de cuál sea su origen (adaptación al contexto, gestión dinámica de la QoS o fallo de nodo).

2.3.1 Adaptación al Contexto (Adaptabilidad)

Hay varios aspectos importantes cuando se trabaja la sensibilidad al contexto: qué se entiende por contexto, qué cambios se consideran relevantes, cómo identificarlos y cómo reaccionar frente a ellos.

En cuanto a la definición del término **contexto**, tal y como se recoge en Baldauf et al. (2007), desde que se introdujo por primera vez en 1994 (Schilit & Theimer, 1994), varios autores le han asignado diferentes significados que comprenden desde la localización de usuarios y su estado emocional hasta los objetos presentes en un entorno. No obstante, la definición más extendida es la proporcionada por Dey: "*cualquier información que se puede usar para caracterizar la situación de entidades (personas, lugares u objetos) que se consideren relevantes para la interacción entre un usuario y una aplicación, incluyendo al propio usuario y a la propia aplicación*" (Dey, 2001, p.5). Además, en Baldauf et al. (2007) también se recoge una clasificación del contexto en función de su dimensión: externa (aquel que puede ser medido mediante sensores hardware como los de localización, iluminación, temperatura, etc.) e interna (aquel especificado por el usuario o basado en sus interacciones, como por ejemplo sus objetivos, tareas ejecutadas o estado emocional). El presente trabajo de investigación se centra en la *dimensión externa* del contexto de las aplicaciones de interés.

A su vez, en la literatura se pueden encontrar diferentes trabajos orientados a la especificación del contexto, tal y como se recoge en el estudio presentado en Strang &

Linnhoff-Popien (2004). Por un lado, algunos identifican características concretas (Hervás et al., 2010; Hoyos et al., 2013). Más concretamente, en Hervás et al. (2010) se propone un modelo formal centrado en el usuario y en el ámbito de la inteligencia ambiental. Basándose en la idea de que el entorno ofrece servicios a los usuarios a través de dispositivos, se proponen cuatro ontologías: modelo de dispositivos (elementos hardware y software), modelo del usuario (perfil, agenda y situación actual), modelo del entorno (organización del espacio físico) y modelo del servicio (qué servicios están disponibles). De forma similar, en Hoyos et al. (2013) se propone un lenguaje para la caracterización de diferentes tipos de contexto: físico (magnitudes físicas), social (leyes, amistades…), computacional (tráfico de red, estado del hardware), entorno (distribución del espacio físico), personal (perfil del usuario: edad, sexo, gustos, etc.) y tareas (qué tareas puede realizar un usuario).

Por otro lado, otros autores proponen aproximaciones más genéricas como Fleurey & Solberg (2009) y Morin et al. (2009) que proponen caracterizar el contexto en base a un conjunto de variables que identifican los elementos del entorno que pueden implicar una adaptación. Estas variables se determinan para cada casuística particular. En el caso de la propuesta realizada en Sheng & Benatallah (2005) se permite relacionar un elemento del contexto con la fuente que proporciona el dato (puede ser el dato original facilitado por un sensor o el dato ya procesado). No obstante, la caracterización del contexto suele realizarse para documentación o para que una plataforma de gestión emplee dicha información para la toma de decisiones relativas a la reconfiguración de las aplicaciones. Para las aplicaciones de interés del presente trabajo, la caracterización del contexto no resulta relevante ya que se considera parte de la funcionalidad de los componentes que constituyen las aplicaciones y que, en última instancia, son los que capturan y/o procesan datos del contexto.

En los siguientes sub-apartados se analizan trabajos relacionados con plataformas de gestión que controlan el proceso de adaptación así como trabajos relacionados con el diseño previo a dicho proceso.

2.3.1.1 Diseño

Las aproximaciones enfocadas en el diseño del proceso de adaptación se pueden dividir en dos grandes grupos: (1) diseño de alto nivel, en ocasiones orientado al experto de dominio y en ocasiones al desarrollador de software, que define la adaptación teniendo en cuenta la funcionalidad de la aplicación, p. ej., detección de incendio representado por un evento o aviso al personal de vigilancia; y (2) diseño de bajo nivel, orientado únicamente a los desarrolladores de software, que se refiere a las operaciones concretas a llevar a cabo durante el proceso de reconfiguración, tales como eliminación, reemplazo o reconexión de módulos.

En el primer grupo, **diseño de alto nivel**, existen trabajos que proporcionan *soluciones cerradas* que se pueden configurar por parte del usuario final. Por ejemplo, en Farella et al. (2010) se permite elegir para varias situaciones relevantes la reacción adecuada, de entre sendas listas predefinidas. En Pérez et al. (2009) se pueden seleccionar los servicios deseados (iluminación, alarma, detección de presencia, etc.) y localizar los dispositivos de medida y actuación correspondientes.

Otros trabajos proporcionan mecanismos para definir la identificación de situaciones relevantes y/o las respuestas apropiadas. Algunos se centran en los *eventos* que representan cambios de contexto. Así, en Søberg et al. (2010) se propone un lenguaje para la descripción de eventos que tiene en cuenta el tipo de sensor, su localización y propiedades temporales, mientras que en Benghazi et al. (2012) se combinan métodos formales y mecanismos de modelado para establecer restricciones en el tiempo de respuesta ante un evento, entendido como el tiempo transcurrido desde que se detecta una situación peligrosa hasta que se activa la alarma correspondiente. Por su parte, Noran (2014) presenta una aproximación que permite establecer relaciones entre eventos, indicando si un evento puede dar lugar a/influenciar en otros. De esta manera se puede determinar si un terremoto puede dar lugar a un tsunami y éste a su vez provocar una pandemia. En el caso de sistemas que presentan gran cantidad de eventos relacionados, como los sistemas de gestión de tráfico, los llamados *Event Processing Language* (EPL) permiten describir las situaciones a

detectar que posteriormente procesa un motor cuyo objetivo es extraer información de valor para los usuarios en base a los eventos producidos (Dunkel et al., 2011). Se trata de lenguajes complejos que requieren de usuarios experimentados, por lo que actualmente existe alguna propuesta que trata de acercar estos lenguajes al experto de dominio, como es el caso de Boubeta-Puig et al. (2014). Se trata de una aproximación basada en modelos y orientada a que los expertos de dominio realicen la definición gráfica de los patrones de eventos. Esta propuesta permite indicar las condiciones que deben cumplir los eventos producidos para generar un nuevo evento más complejo, de forma amigable e independiente de EPLs, para posteriormente generar la definición para un EPL concreto.

También existen aproximaciones que emplean *técnicas de modelado* que permiten *diseñar las aplicaciones junto con las alarmas*. Por ejemplo, en el proyecto MPOWER (Stav et al., 2013) se proporcionan mecanismos para modelar la supervisión de la salud de los pacientes, lo que incluye la detección de situaciones peligrosas. Pero son las enfermeras las que manualmente llevan a cabo las acciones necesarias en caso de alarma. La propuesta descrita en (Rabbi et al., 2014) también se basa en el uso de modelos de forma que las alertas se lanzan como resultado de un procesamiento, para ser visualizadas por diferentes usuarios.

El uso de *reglas* es otro de los mecanismos empleados para el diseño ya que permiten la clasificación de situaciones, como en Botia et al. (2012) y en el sistema CAALYX (Rocha et al., 2013) donde se emplean para detectar situaciones anormales o alarmantes, respectivamente. En el primer caso se reacciona tratando de contactar con la persona afectada y si esto no es posible se lanza una alarma. De forma similar, en CAALYX se identifica y avisa al responsable de llevar a cabo las acciones necesarias, en función de la alarma producida. También basado en reglas, el sistema propuesto en Augusto & Nugent (2004) permite definir una actuación directa sobre el entorno más adecuada para, por ejemplo, encender la luz o apagar la calefacción. En este contexto, el paradigma *Event-Condition-Action* (ECA) permite definir de forma sencilla reglas legibles que se pueden procesar de forma eficaz (Sadri, 2011). Se trata de reglas que presentan el siguiente formato "*On <Evento> If <Condición> Do*

<Acción>" (cuando se produce un *Evento*, si se da una *Condición* se debe ejecutar una *Acción*) e incluso variaciones sobre él como por ejemplo "*On <Evento> Do <Acción>*". A este respecto se han presentado propuestas para reducir la incertidumbre en la especificación de eventos como en Liu et al. (2006) y Augusto et al. (2008).

La Tabla 2-1 recoge un resumen de las aproximaciones agrupadas en "diseño de alto nivel". Como se puede observar, en la mayoría de estos trabajos la respuesta ante un cambio de contexto consiste en el lanzamiento de una alarma, el envío de avisos o la actuación directa sobre el entorno. Pero no se considera la posibilidad de que se defina el reaccionar directamente sobre una actividad de supervisión para arrancarla, detenerla o modificarla.

Tabla 2-1: Diseño de alto nivel de la adaptación al contexto

TRABAJO	IDENTIFICACIÓN DE SUCESOS RELEVANTES	ESPECIFICACIÓN DE REACCIÓN	DISEÑADOR
(Farella et al., 2010)	Seleccionar de una lista.	Seleccionar de una lista.	Experto de dominio
(Pérez et al., 2009)	Seleccionar sensores e indicar su localización.	Seleccionar actuadores e indicar su localización.	Experto de dominio
(Søberg et al., 2010)	Lenguaje formal para descripción de eventos.	No se especifica.	Desarrollador de software
(Benghazi et al., 2012)	Restricciones temporales para la clasificación de situaciones relevantes.	Restricciones temporales para la atención de un evento.	Desarrollador de software
(Noran, 2014)	Establecer relaciones entre eventos.	No se especifica.	Experto de dominio
(Dunkel et al., 2011)	*Event Processing Language* (EPL)	Extracción de información relevante para cambio de ruta.	Desarrollador de software
(Boubeta-Puig et al., 2014)b	*Event Processing Language* (EPL)	No. Aviso de identificación de situaciones relevantes.	Experto de dominio
(Stav et al., 2013)	Modelado de la detección de situaciones peligrosas.	No. Aviso y ejecución manual de las acciones necesarias.	Desarrollador de software (apoyado en el experto de dominio)
(Rabbi et al., 2014)	Modelado de de la detección de situaciones peligrosas.	No. Lanzamiento y visualización de alertas	Desarrollador de software (apoyado en el experto de dominio)

TRABAJO	IDENTIFICACIÓN DE SUCESOS RELEVANTES	ESPECIFICACIÓN DE REACCIÓN	DISEÑADOR
(Botia et al., 2012)	Reglas para clasificación de situaciones: normales / anormales.	No. Contactar con persona afectada o lanzamiento de alarma.	Desarrollador de software
(Rocha et al., 2013)	Reglas para identificación de situaciones alarmantes.	No. Aviso al responsable.	Desarrollador de software (apoyado en el experto de dominio)
(Augusto & Nugent, 2004)	Reglas ECA	Reglas ECA. Actuación directa sobre el entorno.	Desarrollador de software

En lo que respecta al **diseño de bajo nivel**, la adaptación al contexto suele representarse por medio de la *variabilidad* de un sistema (Galster et al., 2014). La variabilidad hace referencia a la posibilidad de cambio en un producto o sistema, de manera que los posibles cambios se deben definir de forma explícita (Aiello et al., 2010). En este caso, el desarrollador de software identifica aquellos elementos en donde puede ocurrir un cambio, los llamados puntos de variación, así como las diferentes variantes posibles, tal y como se hace en Bencomo et al. (2008) y Morin et al. (2009) para aplicaciones basadas en componentes, o en Sun et al. (2010) para servicios web. La aproximación basada en modelos propuesta en Fleurey & Solberg (2009) va un paso más allá y permite especificar la lógica de adaptación de sistemas de gran tamaño relacionando los puntos de variación (y sus variantes) con el contexto, mediante el uso combinado del paradigma ECA y de reglas basadas en propiedades que se refieren a aquellos aspectos del sistema que se deben optimizar. En Khan et al. (2008) la lógica de adaptación se genera a partir de un modelo en el que cada componente de una aplicación es un punto de variación y cada una de sus implementaciones una variante, pudiéndose establecer relaciones entre las variantes de diferentes componentes para situaciones concretas del contexto.

Otros trabajos están enfocados a la definición de *estrategias de adaptación* que comprenden el conjunto de operaciones a realizar sobre elementos de la arquitectura. Así, el lenguaje Stitch (Cheng & Garlan, 2012) permite al desarrollador definir dichas estrategias en forma de árboles de decisión, de manera que a cada operación se le

asocia una probabilidad y un coste, lo que permite seleccionar la estrategia más adecuada. Otra forma de representar estas estrategias es mediante los llamados contratos de adaptación que contienen las condiciones bajo las cuales se debe realizar una reconfiguración. Más concretamente, se trata de sentencias condicionales que determinan las operaciones de reconfiguración necesarias en base a la ocurrencia de determinados eventos (Dowling & Cahill, 2001). Como se ha comentado anteriormente, en Foster et al. (2011) se entiende el proceso de adaptación como un cambio de modo , por lo que se propone una aproximación basada en modelos para que el desarrollador especifique qué situaciones provocan el cambio de un modo a otro, posteriormente a la definición funcional de cada modo. Por último, a muy bajo nivel, hay aproximaciones que se basan en el diseño de un plan de reconfiguración en formato de pseudo-código (Hammer & Knapp, 2010; Hofmeister, 1998; Léger et al., 2010). Por ejemplo, en Hofmeister (1998) mediante un fichero de comandos (*script*) se definen todos los pasos a seguir en el proceso de adaptación: añadir, eliminar, reemplazar, migrar, llegando al detalle de gestión del estado.

Por lo tanto, se trata de trabajos que, en general, exigen un conocimiento detallado de la plataforma de gestión y/o tecnología empleados. Sólo aquellos que diseñan la variabilidad en base a modelos presentan cierto nivel de abstracción, pero en cualquier caso, son soluciones totalmente enfocadas al desarrollador de software.

2.3.1.2 Gestión en Ejecución

En tiempo de ejecución se proponen aproximaciones basadas en **plataformas de gestión**, construidas sobre una arquitectura software, que controlan y llevan a cabo el proceso de adaptación. A continuación se analizan trabajos de diferentes autores, cuyo resumen se muestra en la Tabla 2-2.

Tabla 2-2: Reconfiguración dinámica para adaptación al contexto

TRABAJOS	¿QUIÉN LA SOLICITA?	¿QUÉ ACCIONES SE PERMITEN?
THOMAS (Bajo et al., 2010)	Agentes con determinados roles.	Añadir estructura organizativa. Agregar o eliminar miembros de una estructura organizativa.
DRS (Wegdam et al., 2003)	Diseñador de la solicitud (usuario).	Creación o destrucción de componentes.
(Hofmeister, 1998)	Usuario.	Acciones sobre componentes: reconectar, añadir, eliminar, etc.
(Léger et al., 2010)	Usuario.	Acciones sobre componentes: reconectar, añadir, eliminar, etc.
iLAND (García-Valls et al., 2013b)	Plataforma de gestión o un servicio.	Arrancar y detener aplicaciones o servicios.
ACCADA (Gui et al., 2011) (Khan et al., 2008)	Plataforma de gestión.	Modificar la arquitectura de una aplicación: añadir, eliminar, sustituir y reconectar componentes.

La arquitectura THOMAS (Bajo et al., 2010) combina la tecnología multiagente con la orientación a servicios, permitiendo establecer estructuras organizativas para la colaboración entre agentes. La reconfiguración dinámica del sistema es posible mediante la incorporación de nuevas estructuras organizativas, o mediante la agregación o eliminación de sus miembros. Sin embargo, estas capacidades están restringidas a determinados roles. En Wegdam et al. (2003) se propone un Servicio de Reconfiguración Dinámica (*Dynamic Reconfiguration Service*, DRS) que permite la creación o destrucción de componentes, siendo su objetivo fundamental el llevar a cabo una reconfiguración correcta (asegurar la integridad estructural del sistema así como la consistencia de estados) y transparente para el desarrollador, pero bajo petición del llamado "diseñador de la solicitud". Otras plataformas de gestión también inician la reconfiguración bajo petición de un usuario externo a la aplicación, como por ejemplo en Hofmeister (1998) y Léger et al. (2010).

Por otro lado, algunos trabajos entienden la reconfiguración como *recomposición de las aplicaciones*. Tal es el caso del middleware iLAND (García-Valls et al., 2013b) con un proceso de recomposición acotado en tiempo que puede ser solicitado por un servicio gracias a la interfaz de programación de aplicaciones (*Application*

Programming Interface, API) que el middleware ofrece. En este grupo también se engloba el framework ACCADA (Gui et al., 2011) que inicia el proceso de reconfiguración dinámica debido a dos circunstancias: (1) cuando se detecta que un usuario ha desactivado un componente y se debe reemplazar por otro; o (2) cuando se violan determinadas condiciones establecidas sobre el contexto en fase de diseño. Para ello, ACCADA dispone de un módulo llamado *EventMonitor* encargado de supervisar cambios en el estado de los componentes que indiquen la violación de dichas condiciones. Por último, la lógica de adaptación generada para la plataforma de gestión propuesta en Khan et al. (2008) relaciona valores de características del contexto con implementaciones de componentes. Esta información es usada por dicha plataforma de gestión en tiempo de ejecución para reconfigurar los componentes (elegir nuevas implementaciones) en base al cambio detectado en las condiciones del contexto.

Como se puede observar la Tabla 2-2 ninguna plataforma de gestión posibilita que sea la propia aplicación la que inicie el proceso de adaptación en base a los resultados de su funcionalidad, requisito fundamental de las aplicaciones sensibles al contexto. De hecho, en la mayoría de las plataformas de gestión analizadas el proceso de adaptación lo inicia la propia plataforma de gestión o un usuario externo a la aplicación. Únicamente la plataforma THOMAS permite que determinados agentes soliciten una reconfiguración dinámica para modificar la composición de estructuras organizativas.

2.3.2 QoS Específica de Aplicación

Este apartado se centra en aquellos requisitos que caracterizan la **calidad de servicio** de las aplicaciones, es decir, la *calidad del resultado percibido*. En este punto cabe destacar que generalmente se trata de características que varían de un campo de aplicación a otro, o incluso de una aplicación a otra. Es decir, se trata de una QoS específica de aplicación. A este respecto hay trabajos que proponen mecanismos para la definición de la QoS, otros se centran en su gestión en tiempo de ejecución, y por

último los hay que combinan ambos aspectos. A lo largo de los siguientes párrafos se analizan propuestas de diferentes autores.

Por un lado existen aproximaciones *genéricas* en las que no se indica ninguna característica particular como en Li (2012). Por otro lado, se encuentran trabajos que permiten definir la *QoS percibida por el usuario final*, ligándose a características concretas. Este es el caso de Chalmers & Sloman (1999) y Buccafurri et al. (2008) para aplicaciones multimedia donde se relaciona la QoS con la calidad del contenido ofrecido (video, imagen o sonido). Esta QoS puede caracterizarse, por ejemplo, por la resolución de la imagen o la velocidad del video. Por último, hay aproximaciones que definen la QoS a más bajo nivel en base a *parámetros relacionados con la ejecución*, lo que implica que en ocasiones dicha definición no pueda llevarla a cabo un experto de dominio. Este es el caso de Egbogah & Fapojuwo (2011) donde se determina en términos de su confianza, latencia y eficiencia energética, o el caso de MOSES (Cardellini et al., 2012) donde se relaciona con el tiempo de respuesta de una tarea, su coste computacional y la probabilidad de completar su ejecución.

Para la captura de la QoS específica de aplicación, una opción es recoger su **caracterización** en forma de *anotaciones* sobre el diseño arquitectónico de un sistema. En el proyecto Pecos (Wuyts & Ducasse, 2001), centrado en sistemas empotrados, el modelo de componentes se anota con los requisitos temporales, de planificación y consumo de memoria de los componentes. La propuesta realizada por Sentilles et al. (2009) es más genérica y aborda la definición de cualquier QoS como un atributo caracterizado por su nombre, valor e información acerca de la validez de dicho atributo, incluyendo su relación con otros o sus dependencias con respecto a la plataforma. En este contexto se encuentran aproximaciones que expresan dichas anotaciones mediante perfiles UML: QFTP (OMG, 2008) es un perfil UML de propósito general que se emplea para el modelado de QoS y tolerancia a fallos, pudiendo ser extendido como se propone en Espinoza et al. (2006) para realizar análisis cuantitativos; SPTP (OMG, 2005) es otro perfil UML, pero centrado en sistemas de tiempo real; de la combinación y evolución de QFTP y SPTP se propone MARTE (OMG, 2011) para sistemas empotrados de tiempo real que también introduce la posibilidad

de especificar propiedades cuantitativas. Así, en el proyecto MultiPartes (Salazar et al., 2014; Salazar et al., 2013), enfocado en aplicaciones de criticidad mixta, se emplea el perfil UML MARTE para anotar los modelos de aplicación, que inicialmente contienen su descripción funcional, con información relativa a requisitos temporales tales como su patrón de activación, plazo y tiempo de cómputo. Por su parte, la metodología de modelado MAST-2 (Harbour et al., 2002) para el diseño y análisis de sistemas de tiempo real mono-procesador o distribuidos, también se inspira en el perfil UML MARTE añadiendo nuevos elementos como los conmutadores de red. Una variante del formato de anotaciones se introduce en Wehrmeister et al. (2014) donde se propone representar los requisitos funcionales mediante casos de uso y definir la QoS en base a plantillas, relacionando ambos tipos de requisitos mediante una tabla.

Por otro lado, existen *lenguajes* para la definición de QoS como QML (Frølund & Koistinen, 1999) y QDL (Loyall et al., 1998). El lenguaje de propósito general QML permite agrupar diferentes parámetros de QoS, dimensiones, en los llamados tipos de contratos indicando el tipo de valor asociado a cada dimensión. Un contrato se corresponde con una instancia concreta de un tipo de contrato en la que se especifican valores concretos para cada una de sus dimensiones, asociando cada contrato con un elemento de la arquitectura (interfaz, operaciones o argumentos de operaciones). Mediante QDL se puede establecer la QoS deseada, la QoS proporcionada y posibles niveles aceptados.

Como se puede observar, cada una de estas aproximaciones utiliza diferentes conceptos, métricas, unidades y valores para representar información acerca de la QoS. Pero todas ellas coinciden en definirla en base a varios parámetros o características, y algunas incluso permiten determinar diferentes niveles de calidad aceptables.

En tiempo de ejecución, se emplean **plataformas de gestión** cuyo cometido es asegurar que la QoS esperada se pueda proporcionar. Este hecho está directamente relacionado con la gestión de los recursos disponibles en la infraestructura. Por otro lado, se distinguen dos tipos de situaciones. Aquéllas en las que no se toleran

incumplimientos de QoS, por ejemplo debido a las consecuencias catastróficas que esto pudiera tener como ocurre con las aplicaciones de tiempo real crítico en el ámbito de la aviónica. Y situaciones en las que sí que se tolera cierta relajación de la QoS (p. ej., determinadas aplicaciones multimedia). Dentro del primer tipo se engloba la plataforma FTT-MA (Noguero et al., 2013) que asegura la sincronización temporal de aplicaciones distribuidas cuyos componentes presentan restricciones temporales, definiendo réplicas de instancias en ejecución alojadas en diferentes nodos. De manera que si la disponibilidad de los recursos del sistema cambia, se modifica la planificación de las aplicaciones en base a un criterio establecido en diseño por parte del desarrollador (p. ej., elegir nodo con la mayor capacidad de procesamiento o balanceo del nivel de batería). En el segundo tipo, teniendo en cuenta que las violaciones de QoS suelen deberse a limitaciones en la disponibilidad de recursos (Li & Nahrstedt, 1999), es posible una **gestión dinámica de la QoS**, ajustando la demanda de recursos de estos componentes a los recursos del sistema disponibles.

Esta gestión dinámica de la QoS tiene varias implicaciones ya que se exigen mecanismos que, por un lado, permitan asegurar que la QoS especificada se puede alcanzar, y que, por otro lado, permitan mantener dichos niveles de QoS a lo largo del tiempo (Chalmers & Sloman, 1999; García-Valls et al., 2012). Por lo tanto, las plataformas de gestión deben disponer de mecanismos de control de admisión, de gestión de la reserva de recursos, de monitorización de recursos del sistema y de monitorización del cumplimiento de la QoS. También son necesarios mecanismos de reconfiguración dinámica que permitan adaptar las aplicaciones o componentes para el aseguramiento de su propia QoS o de la QoS de otros elementos.

Para poder llevar a cabo estas operaciones las plataformas de gestión deben tener conocimiento de la **demanda de recursos** de los diferentes componentes. Uno de los mecanismos empleados para ello es el uso de *contratos de QoS* que permiten especificar los niveles de calidad esperados en diferentes condiciones del contexto. En concreto, en Tamura et al. (2014) se propone la definición de contratos mediante máquinas de estado finitas. Cada estado representa un nivel de QoS esperado, mientras que las transiciones entre estados se corresponden con eventos derivados

de cambios en las condiciones del sistema. Cada transición lleva asociada la correspondiente regla de reconfiguración, es decir, el conjunto de operaciones a llevar a cabo por parte de la plataforma denominada QoS-CARE. Por su parte, dentro del proyecto Quadrantis (Tran et al., 2009) se propone gestionar la QoS de aplicaciones basadas en servicios teniendo en cuenta no sólo la QoS relativa a los servicios web sino también la relativa al contenido ofrecido, que en este caso es multimedia (texto, video, sonido, imágenes), así como las preferencias del usuario web. El valor de los parámetros que caracterizan la QoS de los servicios web se personaliza mediante contratos establecidos entre el cliente y el servidor, determinándose en ellos los niveles de QoS aceptables. La QoS del contenido se almacena en perfiles de usuario y consta de parámetros que representan sus preferencias, necesidades y demanda de recursos de red. También para aplicaciones multimedia, pero en sistemas empotrados, en García-Valls et al. (2012) se describe un mecanismo de asignación de prioridades a tareas que junto con mecanismos de asignación dinámica de recursos mejora el uso del procesador por parte de las aplicaciones, permitiendo que algunas de ellas aumenten su consumo de recursos sin que ello afecte al resto. Para lograrlo, se propone desarrollar las aplicaciones de manera que sean capaces de ejecutarse en diferentes modos (niveles de QoS), expresando su demanda de recursos en forma de contratos.

En la literatura se encuentran también aproximaciones no basadas en contratos en las que se indican *propiedades que hacen referencia a la demanda de recursos* de los diferentes elementos del sistema. Así el middleware iLAND (García-Valls et al., 2014) resuelve el problema de la selección de las implementaciones más adecuadas en base a su consumo de CPU, periodo y plazo, teniendo en cuenta que su composición debe cumplir un determinado tiempo de respuesta al mismo tiempo que se debe reducir al mínimo el factor de utilización de los nodos implicados. Por su parte, el framework ACCADA (Gui et al., 2011) tiene en cuenta factores relativos a la CPU como por ejemplo el consumo o tipo de CPU que demandan los componentes. De forma similar, la plataforma de gestión DJINN (Mitchell et al., 1999) considera el consumo de CPU, memoria y ancho de banda de aplicaciones multimedia. En este caso, la demanda de recursos de un componente se completa con un conjunto de reglas que permiten a la

plataforma DJINN comprobar la validez de la nueva configuración. En la arquitectura descrita en Kon et al. (2005) cada componente lleva asociado un conjunto de pre-requisitos, donde se incluye su demanda de recursos (tipo y cantidad de recurso hardware demandado), en los que su correspondiente configurador se basa para realizar la adaptación del nivel de QoS. Por último, la caracterización de la demanda de recursos propuesta dentro del proyecto MUSIC (Hallsteinsen et al., 2012) no es estática. Por el contrario, se asigna a cada componente una serie de funciones para el cálculo, en tiempo de ejecución, de los recursos necesarios por cada una de sus implementaciones. Estas funciones se basan en el valor de los parámetros de QoS identificados para el componente y en las condiciones del contexto en un instante determinado.

2.3.3 Disponibilidad

La gestión dinámica de la QoS mejora la disponibilidad de un sistema ya que ayuda a resolver situaciones de sobrecarga. Sin embargo, en un escenario de fallo de nodo o de componente se precisan mecanismos que minimicen la interrupción del servicio de las aplicaciones, teniendo en cuenta que la disponibilidad de una aplicación viene determinada por la disponibilidad de todos sus componentes. La gestión de la redundancia es el mecanismo más empleado (Hassine, 2015).

Una solución es soportar la tolerancia a fallos a través de la programación. Tal es el caso de (Gharzouli & Boufaida, 2009) donde se presenta una aproximación descentralizada de composición en la que cada servicio busca un proveedor disponible estableciendo comunicaciones punto a punto entre ellos. En Bloom & Day (1993) se propone que cuando un cliente reciba una excepción como respuesta a la solicitud de un servicio, sea él mismo quien busque otro proveedor disponible. De forma similar, en el proyecto MUSIC (Hallsteinsen et al., 2012) es la propia aplicación la encargada de asegurar la integridad del estado en caso de fallo de nodo. En el sistema MAS propuesto en García-Magariño & Gutiérrez (2013) cuando un agente falla, el resto de agentes en activo se encargan de suplir su baja, asumiendo su funcionalidad.

La redundancia de recursos físicos es otra de las alternativas, sobre todo en sistemas críticos (Qiu et al., 2017) como los sistemas de control de vuelo (Goupil, 2010). Por ejemplo, una de las tácticas empleadas es la de la triple redundancia según la cual tres elementos idénticos realizan un mismo procesamiento, enviando sus resultados a un cuarto que mediante un sistema de votación es capaz de detectar inconsistencias (Hassine, 2015). En el caso de la plataforma GAL (Eichelberg et al., 2010) se opta por disponer de dos canales de comunicación con objeto de asegurar que las notificaciones de alarmas no se pierden.

Todas estas aproximaciones permiten la detección rápida de problemas así como dar la solución más óptima para una aplicación concreta. Sin embargo, se carece de una visión global del sistema, además de supeditar el diseño de la lógica de la aplicación al aseguramiento de su disponibilidad.

Otros trabajos optan por proponer **soluciones transparentes para la aplicación**, basadas en plataformas que gestionan el proceso de recuperación ante un fallo. En este contexto, la arquitectura THOMAS (Bajo et al., 2010) permite proveedores redundantes para un mismo servicio ofrecido, siendo la propia arquitectura la encargada de realizar las búsquedas de proveedores. El framework Gravity (Cervantes & Hall, 2004) propone un entorno de ejecución que gestiona la disponibilidad creando y destruyendo conexiones entre instancias de componentes en base a información sobre sus dependencias recogida en el modelo de componentes. El middleware iLAND (García-Valls, Rodríguez-López, et al., 2013) soporta diferentes implementaciones para un mismo servicio, de manera que la recuperación consiste en la recomposición dinámica de sus implementaciones. De forma similar, la arquitectura FTT-MA (Noguero et al., 2013) gestiona diferentes réplicas de una tarea, entre las cuales sólo una puede estar activa.

2.3.4 Transferencia del Estado

Resulta importante señalar que en ocasiones las entidades funcionales que conforman las aplicaciones (componentes, servicios o agentes) tienen estado, es decir

su ejecución depende del resultado de ejecuciones anteriores, siendo uno de los principales retos de cualquier proceso de reconfiguración dinámica, independientemente del motivo que da lugar a dicha reconfiguración, el asegurar la persistencia de su estado (Li, 2012). En este contexto, hay tres aspectos a tener en cuenta: (1) describir el estado de una entidad; (2) definir la forma de transferir el estado; (3) determinar el instante en el que se realiza la reconfiguración, ya que no sólo el valor del estado sino también su estructura puede variar de un instante a otro. Por ejemplo, tal y como indica Vandewoude (2007) si el proceso de adaptación se lleva a cabo *tras completar un ciclo de ejecución* su estado podría quedar definido por el valor de determinadas variables globales e internas del componente, mientras que si se produce *durante su ejecución* sería necesaria más información como por ejemplo la pila de ejecución.

Con respecto al primer punto, hay trabajos que establecen una **estructura o formato** del estado que habitualmente suele ser *abstracto* para posibilitar su transferencia entre diferentes plataformas (Bloom, 1983; Hammer & Knapp, 2010; Hofmeister, 1998), mientras que otros no determinan *ningún formato concreto*, ofreciendo mecanismos de serialización y dejando en manos del desarrollador la responsabilidad de gestionar estructuras compatibles (Hallsteinsen et al., 2012; Vandewoude et al., 2007).

En cuanto a la **transferencia del estado**, se pueden distinguir dos aproximaciones (Hammer, 2009; Vandewoude, 2007). Se habla de *transferencia directa* cuando la plataforma que gestiona la reconfiguración conoce cómo acceder y actualizar el estado de los bloques funcionales, mientras que en la *transferencia indirecta* es el propio bloque el que implementa los mecanismos para exportar y restaurar su estado, habitualmente mediante métodos *get/set*. A este respecto, aunque hay algún trabajo basado en transferencia directa (Vandewoude, 2007), la mayoría opta por la transferencia indirecta, como en Bloom & Day (1993), Hallsteinsen et al. (2012), Hammer & Knapp (2010), Hofmeister (1998) y Kon et al. (2005). Resulta importante destacar que en el caso de sistemas MAS la transferencia del estado de un agente en

ejecución puede no ser necesaria, debido a su capacidad de migrar de un nodo a otro, viajando el agente junto con su estado.

En la **elección del instante óptimo** para llevar a cabo el proceso de reconfiguración se debe tener en cuenta que durante un reemplazo las dos instancias involucradas no pueden ejecutarse a la vez. Además, en dicho instante el estado de las entidades afectadas debe ser estable. Esto implica que las acciones que conforman el proceso de reconfiguración deben estar coordinadas (Li, 2012). El concepto de *quiescence* (Kramer & Magee, 1990) se introdujo para denominar a este instante óptimo. Este concepto se refiere a una situación en la que un componente software (nodo en su terminología) está pasivo (puede aceptar y servir transacciones, pero no se encuentra involucrado en ninguna transacción iniciada por él, ni iniciará nuevas transacciones), hecho que provoca que todos aquellos nodos con los que tiene conexión también pasen a estar pasivos. Sin embargo, a pesar de que alcanzar una situación de quiescence asegura la consistencia del estado, supone un gran bloqueo de la ejecución de la aplicación. De hecho, Li (2011) presenta un estudio de cómo un proceso de reconfiguración puede afectar a la ejecución de los sistemas, analizando diferentes aproximaciones de diferentes autores. Es por ello que en Vandewoude et al. (2007) se propuso como alternativa la situación de *tranquility* en la que sólo el módulo envuelto en la reconfiguración tiene que ser llevado al estado pasivo. Con este objetivo los autores proponen forzar alcanzar dicha situación bloqueando los mensajes entrantes al componente.

En cualquier caso, el *bloqueo* de parte de los componentes afectados por un proceso de reconfiguración es la práctica más habitual para alcanzar la consistencia en el estado. Por ejemplo, en Hofmeister (1998) se bloquean las comunicaciones con el módulo afectado lo que implica tener que guardar también la cola de mensajes recibidos. Tras la actualización de las conexiones al nuevo módulo se le restaura el estado y la cola de mensajes. Algo similar se propone en Wegdam et al. (2003) donde para alcanzar el estado seguro de reconfiguración además de interrumpir las interacciones, todos los componentes disponen de un mecanismo que los convierte en reactivos (únicamente se activan bajo la recepción de peticiones de otros

componentes). El encolamiento de mensajes de entrada no es necesario en Hammer & Knapp (2010), ya que el nuevo componente se instancia en estado inactivo, redirigiendo todas las comunicaciones de la otra instancia a la nueva. Así, la nueva instancia recibe peticiones pero no las procesa hasta que el proceso de reconfiguración ha finalizado, y la otra instancia ha sido eliminada.

Sin embargo, también existen aproximaciones que abogan por *no bloquear* la ejecución de los componentes. Por ejemplo, para el caso particular de sistemas de tiempo de real, en Cano & García-Valls (2014) se definen tareas de gestión encargadas de las operaciones de reconfiguración (nuevo componente, reemplazo de un componente, desinstalar un componente o reconectar componentes) que se deben planificar con el resto de tareas funcionales. Para ello, se propone medir, fuera de ejecución, los tiempos de carga/descarga/reemplazo de cada componente que permiten planificar dichas tareas para su ejecución en segundo plano, asignándoles una prioridad tal que se eviten efectos no deseados. Como resultado, se logra un proceso de reconfiguración acotado en el tiempo, imprescindible en sistemas de tiempo real.

Por último, en una situación de **fallo de nodo o componente** ni el proceso de reconfiguración ni la transferencia del estado pueden ser anticipadas. Por lo tanto, los mecanismos presentados en este sub-apartado no son aplicables. De hecho, ninguno de los trabajos presentados soporta la recuperación del estado en caso de fallo de nodo, para lo cual es necesario que un elemento externo a la instancia en ejecución guarde su último estado conocido (otra instancia redundante o una plataforma de gestión).

2.3.5 Seguridad

En la literatura se pueden encontrar varios trabajos referidos a la **seguridad y privacidad** del almacenamiento, procesamiento y transmisión de datos. Las soluciones más habituales se corresponden con: encriptación en base a infraestructuras de clave pública (*Public Key Infrastructure*, PKI) (Rocha et al., 2013;

Stav et al., 2013), el uso de *Secure Socket Layer* (SSL) (Corchado et al., 2008), mecanismos de autenticación y autorización (Bajo et al., 2010), establecimiento de redes privadas virtuales (*Virtual Private Network*, VPN) (Büsching et al., 2012) o desarrollo de frameworks (Su & Wu, 2011; Vitabile et al., 2009).

En general, se trata de mecanismos ofrecidos por la propia plataforma de gestión para los cuales no es necesario capturar información en fase de diseño.

2.4 Soporte al Desarrollo

Teniendo en cuenta que las aplicaciones distribuidas sensibles al contexto son complejas, resulta imprescindible disponer de buenas técnicas que mejoren tanto su especificación como su desarrollo. Por lo tanto, en este apartado se analizan trabajos que dan soporte al ciclo de desarrollo de aplicaciones, bien para su diseño bien para su implementación. Más concretamente, se analizan aproximaciones basadas en la ingeniería dirigida por modelos (*Model Driven Engineering*, **MDE**), que considera a los **modelos** (abstracciones simplificadas de la realidad que únicamente contienen información relevante para un determinado receptor o propósito) como elementos cruciales que dirigen el proceso de desarrollo. Así, los modelos se emplean como entradas y salidas de las diferentes fases del ciclo de desarrollo hasta su generación (Balasubramanian et al., 2006; Selic, 2003). De hecho, MDE da soporte a las diferentes actividades de desarrollo mediante el uso de *lenguajes de modelado* y *motores de transformación* (Schmidt, 2006). Además, la automatización de tareas se considera también uno de los pilares de MDE, por lo que cualquier propuesta basada en dicha tecnología debe ir acompañada de su correspondiente **entorno de soporte** (Selic, 2008). Se trata de herramientas que facilitan y automatizan el diseño de los modelos y/o la realización de transformaciones a partir de ellos, para generar código (transformaciones modelo-texto (*model-to-text*, *M2T*)) o para generar nuevos modelos (transformaciones modelo-modelo (*model-to-model*, M2M)).

Se pueden encontrar trabajos que hacen uso de *UML como lenguaje de modelado*. Este es el caso del proyecto MUSIC (Hallsteinsen et al., 2012) que emplea la herramienta

Papyrus (The Eclipse Foundation, 2015) para el modelado de aplicaciones junto con su QoS y demanda de recursos. Para la generación de código se emplea MOFScript (Oldevik et al., 2005), obteniéndose el esqueleto de los componentes así como las declaraciones de las funciones que la plataforma de gestión necesita para el cálculo de los recursos requeridos por las implementaciones de dichos componentes. Otro ejemplo se encuentra en el entorno de soporte MPOWER (Stav et al., 2013; Walderhaug et al., 2007) que propone realizar el diseño de las aplicaciones y de los servicios web necesarios mediante la herramienta Enterprise Architect (Sparx Systems, 2016). De nuevo, se emplea MOFScript para la generación automática de los ficheros WSDL correspondientes a los servicios web y de gran parte del código fuente necesario. Sin embargo, parte de la lógica de operación tiene que ser programada manualmente por el desarrollador en base a los modelos definidos. Por su parte, el entorno de soporte presentado en Sun et al. (2010) está enfocado al diseño de la variabilidad en servicios web para lo que hace uso de la herramienta de código abierto para modelado en UML llamada ArgoUML (ArgoUML, 2001). Menos restrictivas son las aproximaciones presentadas en Wehrmeister et al. (2014) y Khan et al. (2008) donde se aceptan modelos UML obtenidos a través de cualquier herramienta (comercial o de código abierto), en formato *XML Metadata Interchange* (XMI) (OMG, 2015). En el primer caso la generación de código se basa en *scripts*, mientras que en el segundo caso un generador basado en MOFScript permite la obtención de la lógica de adaptación.

Por otro lado, hay trabajos que se apoyan en el proyecto *Eclipse Modeling Project* (EMP) (Gronback, 2009) para dar soporte tanto al diseño como al desarrollo. Se trata de una de las plataformas más utilizadas para el desarrollo de soluciones basadas en MDE, formada por una colección de sub-proyectos entre los que se encuentran los proyectos *Eclipse Modeling Framework* (EMF) (Steinberg et al., 2008) y *Graphical Modeling Framework* (GMF) (Eclipse, 2010a) que facilitan el desarrollo de lenguajes de modelado gráficos. El primero permite definir la estructura del lenguaje de modelado, mientras que el segundo se centra en su representación gráfica. Este es el caso de la herramienta desarrollada dentro del proyecto MultiPARTES para la definición, validación y generación de código en sistemas de criticidad mixta en los

que un hipervisor gestiona las diferentes particiones del sistema (Salazar et al., 2014; Salazar et al., 2013). El modelado comprende la descripción funcional de las aplicaciones y la caracterización de la plataforma, estableciéndose relaciones entre ambas y restricciones para las diferentes particiones. De esta manera un generador de código basado en Acceleo (Obeo Network, 2006) proporciona los ficheros de configuración de dichas particiones así como el esqueleto del código de las aplicaciones. También hay propuestas con editores gráficos amigables desarrollados con GMF, como el framework presentado en Sánchez et al. (2011) para aplicaciones en domótica y con generación de código basada en JET (Eclipse, 2010b). La herramienta FTT-Modeler (Noguero & Calvo, 2012) para aplicaciones distribuidas gestionadas mediante la plataforma FTT-CORBA, una implementación de la arquitectura FTT-MA (Noguero et al., 2013) descrita anteriormente. En este último ejemplo para la generación de código se ha empleado MOFScript. Por último, dentro del proyecto iLAND también se propone una herramienta de modelado llamada MTS (Armentia et al., 2011) que permite la especificación gráfica de las aplicaciones orientadas a servicio y la generación automática del esqueleto del código de los servicios, mediante Acceleo.

Incluso existen aproximaciones que implementan sus propios entornos de soporte sin apoyarse en herramientas UML conocidas o frameworks de desarrollo de entornos, como ocurre con la herramienta Genie (Bencomo et al., 2008) para el diseño y desarrollo de aplicaciones basadas en componentes, cuya reconfiguración se expresa en términos de su variabilidad. Genie permite el diseño de los puntos de variación y sus correspondientes variantes, generando no sólo el código de los componentes sino también la política de reconfiguración definida.

En cualquier caso, el desarrollo de entornos de soporte suele ser una tarea compleja que requiere un elevado nivel de experiencia. Es por ello que se han propuesto iniciativas para facilitarlo como las variantes de GMF desarrolladas dentro de Eclipse: EuGENia (Eclipse Epsilon Project, 2009) o Graphiti (Eclipse Modeling Project, 2014). Más genéricas son las propuestas realizadas en Cuevas et al. (2016a) y (Cuevas et al. (2016b); Cuevas (2016)). En el primer caso se aborda la generación de meta-modelos

correspondientes a vistas de un meta-modelo de dominio, facilitándose la generación automática de dicho meta-modelo de vista, en base a las restricciones sobre el del dominio, y de las transformaciones M2M necesarias para convertir un modelo acorde a la vista en un modelo conforme al meta-modelo de dominio. En el segundo caso se propone un diseño genérico y desarrollo basado en MDSE para entornos de soporte, para lo cual las diferentes herramientas que componen el entorno y sus relaciones se expresan en modelos.

Por último, resulta importante destacar que casi todas las aproximaciones basadas en modelos y herramientas o entornos presentados en este capítulo están orientadas al desarrollador. Basándose en la idea de que el experto de dominio es el principal conocedor de los problemas de un ámbito concreto, en la literatura se pueden encontrar corrientes enfocadas a incluirle en el desarrollo de las aplicaciones. Por ejemplo, en (Rabbi et al., 2014) se propone modelar las diferentes tareas de un proceso de salud siempre con el apoyo de un experto médico. Por otro lado, en Pérez et al. (2009) el desarrollador diseña mientras que los expertos configuran para personalizar el sistema a sus necesidades y expectativas. En algunos casos, la intención es que los expertos participen como co-diseñadores. Tal es el caso de la herramienta MPOWER (Stav et al., 2013) donde el diseño de las aplicaciones consta del diseño del negocio realizado por el experto de dominio y del diseño de los servicios web necesarios, llevado a cabo por el desarrollador de software.

2.5 Conclusiones

En este capítulo se han analizado trabajos relacionados que abordan los requisitos demandados por las aplicaciones distribuidas sensibles al contexto. Más concretamente, se ha centrado en sus demandas de: distribución, sensibilidad al contexto, gestión de la QoS específica de aplicación, disponibilidad en caso de fallo de nodo y seguridad.

Las aproximaciones encontradas en la literatura revelan que en **tiempo de ejecución** es necesaria una **plataforma de gestión** que, basándose en una *arquitectura software*

como las descritas en el apartado 2.2, controle la ejecución de las aplicaciones. Este hecho se debe tener en cuenta desde el diseño ya que para una correcta gestión de las aplicaciones, dichas plataformas de gestión deben disponer de *información de su diseño*. Por lo tanto, a continuación se analiza cada uno de los requisitos indicados, con el fin de identificar lo que hace falta capturar en diseño para que en tiempo de ejecución dicho requisito sea posible. Todo ello teniendo en cuenta que uno de los principales objetivos del presente trabajo consiste en que el experto de dominio sea el que realice la especificación de las aplicaciones. La Tabla 2-3 recoge un resumen de cómo las diferentes plataformas de gestión analizadas cumplen dichos requisitos, atendiendo también a si disponen de soporte al desarrollo.

Tabla 2-3: Resumen de plataformas de gestión y cumplimiento de requisitos

PLATAFORMA	DISTRIBUCIÓN	ADAPTABILIDAD	QoS ESPECÍFICA DE APLICACIÓN	DISPONIBILIDAD	CONSITENCIA DEL ESTADO	SEGURIDAD	SOPORTE AL DESARROLLO
THOMAS (Bajo et al., 2010)	SI Agentes y servicios.	SI Restringida a determinados roles.	NO	SI Transparente a la aplicación. Proveedores de servicio redundantes.	NO	SI Autenticación y autorización.	NO
DRS (Wegdam et al., 2003)	SI Componentes.	SI Bajo petición de un usuario externo a la aplicación.	NO	NO	SI Transferencia indirecta.	NO	NO
(Hofmeister, 1998)	SI Componentes.	SI Comandos de muy bajo nivel, incluyendo la localización del código ejecutable.	NO	NO	SI Transferencia indirecta	NO	NO
(Léger et al., 2010)	SI Componentes.	SI Pseudo-código en ficheros de comandos.	NO	SI Sólo durante el proceso de reconfiguración, gracias a los mecanismos de marcha atrás.	SI Dispone de mecanismos de marcha atrás.	NO	NO
iLAND (García-Valls et al., 2013b)	SI Servicios.	SI Proporciona API para la reconfiguración acotada en el tiempo de servicios y aplicaciones	SI Cumplimiento del tiempo de respuesta extremo a extremo, minimizando el factor de utilización de los nodos.	SI Transparente a la aplicación. Diferentes implementaciones de un servicio.	NO	NO	SI Diseño y generación del esqueleto del código.
ACCADA (Gui et al., 2011)	SI Componentes.	SI La plataforma de gestión es totalmente consciente del contexto.	SI Elegir la implementación más adecuada	NO	SI	NO	NO
(Khan et al., 2008)	SI Componentes.	SI La plataforma de gestión se encargada de detectar cambios en las condiciones del contexto.	NO	NO	NO	NO	SI Diseño y generación de la lógica de adaptación.

PLATAFORMA	DISTRIBUCIÓN	ADAPTABILIDAD	QoS ESPECÍFICA DE APLICACIÓN	DISPONIBILIDAD	CONSITENCIA DEL ESTADO	SEGURIDAD	SOPORTE AL DESARROLLO
MOSES (Cardellini et al., 2012)	SI Servicios.	NO	SI Elegir la implementación más adecuada.	SI Gestión de réplicas, excepto en servicios con estado.	NO	NO	NO
QoS-CARE (Tamura et al., 2014)	SI Componentes.	NO	SI Cumplimiento de contratos de QoS asociados a componentes.	NO	NO	NO	NO
DJINN (Mitchell et al., 1999)	SI Componentes.	NO	SI Componente junto con reglas para validar su nueva configuración.	NO	NO	NO	NO
(Kon et al., 2005)	SI Componentes.	NO Dispone de un API para solicitar la ejecución de una nueva aplicación.	SI Mejor localización para ejecutar una aplicación.	NO	SI Transferencia indirecta	NO	NO
MUSIC (Hallsteinsen et al., 2012)	SI Componentes.	NO	SI La demanda de recursos de un componente se calcula en tiempo de ejecución.	SI	SI No transparente a la aplicación.	NO	SI Diseño y generación de código
(García-Magariño & Gutiérrez, 2013)	SI Agentes.	NO	NO	SI No es transparente a la aplicación.	NO	NO	SI Basado en *Ingenias Development Kit*
Gravity (Cervantes & Hall, 2004)	SI Componentes.	NO	NO	SI Transparente a la aplicación. Selección de instancias en repositorio	NO	NO	NO
FTT-MA (Noguero & Calvo, 2012)	SI Componentes.	NO	SI Cambia las instancias para asegurar la sincronización de las aplicaciones.	SI Gestión de réplicas de instancias y dispositivos.	NO	NO	SI Diseño, generación de código y monitorización.

Con respecto a los requisitos de **distribución**, **personalización** y **escalabilidad**, las arquitecturas software presentadas tienen en común el considerar a las aplicaciones como conjuntos de módulos o entidades interconectadas, para las cuales es necesario *definir su configuración*: qué es una entidad, cómo interactúa con las demás y sus conexiones con otras entidades. A este respecto se debe tener en cuenta que un mismo módulo pueda ser empleado en diferentes aplicaciones y que dos módulos sólo pueden interactuar si ambos están de acuerdo en lo que cada uno proporciona o requiere. Sin embargo, muchas de las aproximaciones presentadas no consideran las interconexiones entre entidades desde un punto de vista global, centrándose en la definición de los mensajes intercambiados entre ellas. En este punto cabe destacar que la aproximación propuesta en el presente trabajo se basa en el modelo de servicio desarrollado en el marco del proyecto iLAND (García-Valls, Rodríguez-López, et al., 2013), en el que tanto la autora como las directoras de tesis participaron. Más concretamente, en iLAND se considera un servicio como una pieza de software que recibe datos a través de una interfaz de entrada y los procesa generando un resultado que se entrega a otros servicios a través de su interfaz de salida. Por último, un servicio puede ser realizado por varias implementaciones de servicio, que en última instancia son las unidades de ejecución.

En cuanto a la **adaptación al contexto**, la *dimensión externa* del contexto de las aplicaciones de interés se corresponde con el *entorno bajo supervisión* (espacio físico o persona a supervisar). De esta manera, al ser las aplicaciones las que interactúan con su entorno, a través de sensores gestionados por alguno de sus componentes, son ellas las que deben solicitar el proceso de adaptación. Siendo *innecesaria*, por tanto, una *representación concreta de dicho contexto*. Sin embargo, tal y como se muestra en la Tabla 2-2, pocas plataformas de gestión soportan reconfiguraciones solicitadas por una aplicación. Además, casi ninguna permite actuar sobre el conjunto de una aplicación, centrándose en acciones sobre componentes. Como consecuencia, este hecho tampoco es contemplado por ninguno de los trabajos relacionados con el diseño de la adaptabilidad. En este sentido, teniendo en cuenta que es el experto de dominio el que debe capturar las necesidades de adaptación al contexto de las aplicaciones, las propuestas clasificadas como “diseño de bajo nivel” no se consideran

apropiadas, ya que no permiten separar la definición de la aplicación de las particularidades de la plataforma. De hecho, lograr la *separación de aspectos* necesaria para abstraer al experto de dominio de los aspectos de la implementación es uno de los principales objetivos del presente trabajo de investigación. En las aproximaciones comprendidas dentro del grupo "diseño de alto nivel" hay muchos trabajos que, aunque tengan en cuenta o se basen en el conocimiento del experto, están orientados al desarrollador. Además, la mayoría de ellos reducen su respuesta al lanzamiento de una alarma o envío/visualización de avisos. Se detecta, por lo tanto, que hasta donde la autora conoce, no existe una solución lo suficientemente flexible que permita *identificar sucesos relevantes y definir la reacción frente a ellos*, de manera que dicha reacción pueda ser *solicitada por la propia aplicación* en tiempo de ejecución y contemple el *actuar sobre otras aplicaciones*, sin la intervención de un usuario.

Los trabajos relacionados con la **gestión de la QoS específica de aplicación** muestran que una plataforma de gestión debe conocer la *demanda de recursos de los componentes*, en especial si se trata de gestión dinámica de la QoS. De hecho, esta es la información que dicha plataforma de gestión emplea para control de admisión, para balanceo de carga, para gestionar dinámicamente la QoS, etc. Sin embargo, se trata de una caracterización poco relevante desde el punto de vista del experto de dominio que demanda una *descripción de la QoS relacionada con su área de conocimiento* o con los datos procesados por la aplicación. Es por ello, que ambas representaciones deben ser capturadas en la especificación de las aplicaciones, aunque por parte de diferentes usuarios. Con respecto a la caracterización realizada por el experto de dominio, los conceptos introducidos por los lenguajes QML (Frølund & Koistinen, 1999) y QDL (Loyall et al., 1998) resultan interesantes. Por un lado, se trata de lenguajes genéricos que no se ligan a un campo o clasificación de QoS concretos. Por otro lado, introducen la idea de QoS representada por agrupaciones de parámetros y niveles de flexibilidad. Sin embargo, se trata de soluciones de muy bajo nivel con una sintaxis similar a la de un lenguaje de programación.

Por último, en lo que respecta a la **disponibilidad** de las aplicaciones, incluir la recuperación frente a fallo en la funcionalidad de un componente puede proporcionar la solución más óptima en casos concretos. Sin embargo, lo cierto es que condiciona totalmente la definición de la lógica de la aplicación, reduciendo la reutilización de sus componentes. Por otro lado, con respecto a la *recuperación transparente a la aplicación*, es necesario que la plataforma de gestión proporcione los mecanismos necesarios para que un elemento externo a la instancia en ejecución guarde su último estado conocido, para que pueda ser restaurado en cualquier nueva instancia. En cualquier caso, se trata de un requisito cuyo único responsable es la plataforma de gestión, por lo que *no afecta a la definición de la aplicación*.

En los siguientes capítulos se presenta una propuesta para *capturar*, desde las primeras etapas del diseño, toda la *información necesaria para que*, en tiempo de ejecución, *se puedan cumplir los requisitos de las aplicaciones de interés*. Se trata de una propuesta genérica, independiente de la plataforma de gestión y dirigida a los diferentes usuarios que participan en el ciclo de desarrollo de las aplicaciones. Para lograr la separación de conceptos se propone una aproximación basada en **MDE**, ya que esta tecnología proporciona el nivel de abstracción necesario para la especificación de sistemas complejos. También se presentan los mecanismos y herramientas para dar soporte al ciclo de desarrollo de las aplicaciones, guiando su especificación y automatizando, hasta donde sea posible, su generación de código.

3 APROXIMACIÓN DE MODELADO: CADAMM

"Para abrir nuevos caminos, hay que inventar, experimentar, crecer, correr riesgos, romper las reglas, equivocarse… y divertirse."

(Mary Lou Cook)

3.1 Introducción

En este capítulo se propone una **solución basada en modelos** que proporciona los mecanismos necesarios para **capturar** la **especificación y diseño** de aplicaciones distribuidas sensibles al contexto (a partir de ahora aplicaciones). Por un lado, la aproximación de modelado propuesta recoge toda la información necesaria para que una plataforma de gestión pueda cumplir con los requisitos de dichas aplicaciones en ejecución. Por otro lado, es la base del entorno integrado propuesto en el Capítulo 4 para dar soporte a su ciclo de desarrollo. Por lo tanto, la aproximación de modelado también captura toda la información necesaria para generar las aplicaciones, desde el diseño de alto nivel hasta el detalle necesario para generar su código. Además, lo hace abstrayendo a los diferentes actores de la información de otros dominios.

El ciclo de desarrollo de las aplicaciones se muestra en la Figura 2-1, en la que todos los elementos objeto del presente trabajo de investigación se han resaltado en color azul. En esta figura se observa que el **experto de dominio** es el actor fundamental en la fase de **diseño arquitectónico** ya que es el que conoce con detalle la problemática de su ámbito. En este contexto, la aproximación de modelado propone mecanismos para capturar la información relativa a la *especificación de las aplicaciones*, de forma independiente de la tecnología subyacente y plataforma de gestión. Además, el experto de dominio también conoce el entorno físico (dispositivos que captan información del entorno y dispositivos que actúan sobre él). Por ejemplo, en el caso de una residencia de ancianos, las diferentes plantas y las habitaciones asignadas a cada paciente, así como la ubicación de los sensores que miden sus señales biomédicas (pulsioxímetros, glucómetros, etc.). De la misma manera, para definir la detección de incendios en un monte, los ingenieros forestales conocen sus diferentes regiones y la ubicación concreta de los sensores de CO_2 y de temperatura en cada una de ellas. Por lo tanto, se puede deducir que el modelo que lo representa, el *modelo del entorno físico*, es fijo y conocido, y totalmente dependiente del campo de aplicación.

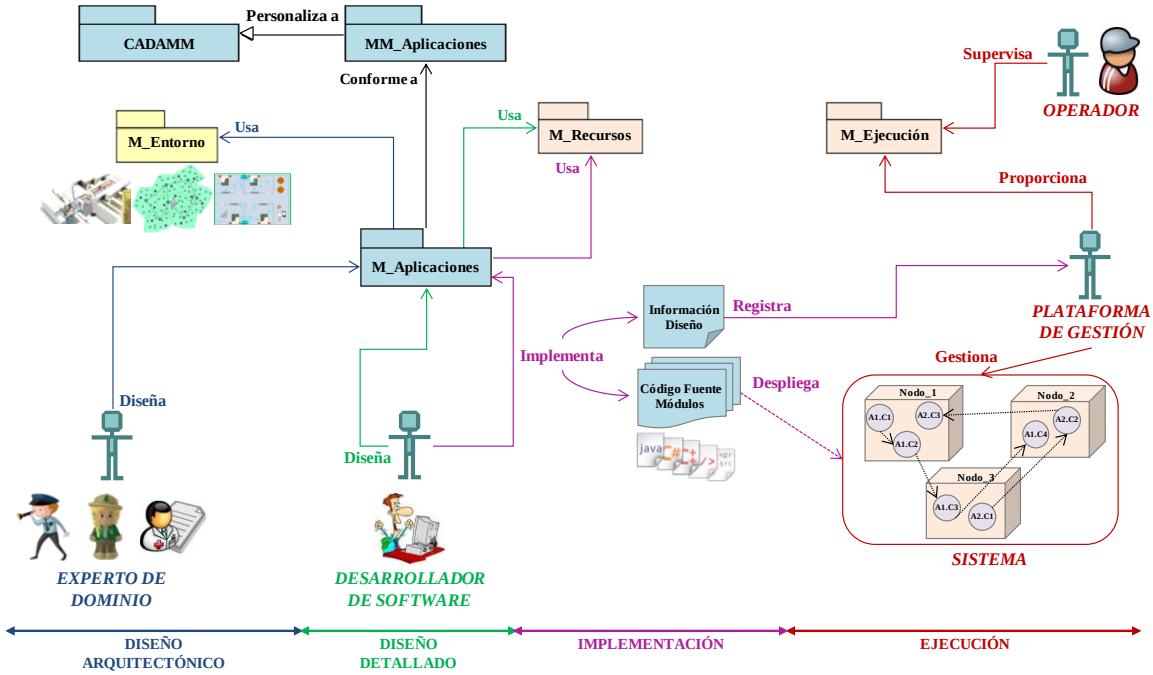


Figura 3-1: Ciclo de desarrollo de aplicaciones distribuidas sensibles al contexto

En la siguiente fase, el **desarrollador de software** (desarrollador a partir de ahora) realiza el **diseño detallado** de la solución SW, basándose en el diseño arquitectónico del experto de dominio. Se identifica la funcionalidad que debe proveer cada parte de la aplicación, incluyendo acciones de inicialización o finalización (por ejemplo de una cámara de video, de un sensor, de un algoritmo de procesamiento…), y la necesidad de mantener o no el estado de las aplicaciones. Es importante destacar que se trata de un diseño aún independiente de la tecnología, ya que no se liga a ningún lenguaje de programación ni plataforma de gestión. Es en la siguiente fase de **implementación** en la que el desarrollador de SW desarrolla el *código* de los diferentes módulos de las aplicaciones (teniendo en cuenta la plataforma sobre la que se ejecutará dicho código), que posteriormente será desplegado en distintos nodos del sistema. Cabe destacar que el desarrollador en ambas fases conoce los *recursos de la infraestructura* y puede establecer las necesidades de recursos de sus módulos. Fundamentalmente, se trata de restricciones a plataforma hardware (p. ej., un código que gestione un sensor únicamente se puede ejecutar en el equipo que tiene acceso a dicho sensor), aunque también pueden ser a plataforma SW (p. ej., un código que procese imágenes puede necesitar unas determinadas librerías que únicamente se encuentran disponibles en equipos concretos). El modelo que caracteriza los recursos del

sistema, *M_Recursos*, es dinámico ya que los recursos pueden darse de alta o de baja en el sistema a lo largo del tiempo.

Por último, en **fase de ejecución** se observa que el *Sistema* está formado por todas las aplicaciones que están en ejecución y los recursos que utilizan o pueden utilizar. Existe también una plataforma que gestiona tanto la ejecución de las aplicaciones como los recursos de la infraestructura. Más concretamente, esta plataforma debe velar por que las aplicaciones se ejecuten tal y como el experto de dominio ha establecido en el diseño arquitectónico, gestionando los recursos del sistema. Además, en caso de que ocurra un fallo en un equipo se debe encargar de recuperar las actividades de supervisión afectadas de forma transparente. Para poder realizar esta gestión y para la toma de decisiones en caso de situaciones alarmantes, la plataforma de gestión necesita conocer información sobre el diseño de las aplicaciones y sus requisitos de ejecución antes de su puesta en marcha (proceso de *registro*). Por último, la plataforma de gestión debe informar sobre el estado de ejecución en el tiempo (representado por el modelo dinámico *M_Ejecución*). De esta manera, un operador que supervise dicho modelo podrá, por ejemplo, saber si un equipo está en fallo y ha quedado fuera de servicio.

Los modelos identificados se definen conforme a meta-modelos que establecen los conceptos empleados, las relaciones entre ellos y las reglas que determinan cuándo un modelo está correctamente formado (Schmidt, 2006). En concreto, el modelo M*_Aplicaciones* de la Figura 2-1 recoge el diseño de las aplicaciones, para cuya especificación se necesitan el modelo de recursos (*M_Recursos*) y el del entorno físico (*M_Entorno*). Este modelo *M_Aplicaciones* es conforme al meta-modelo *MM_Aplicaciones*, que corresponde a una personalización del meta-modelo **CADAMM** (**C**ontext-**A**ware **D**istributed **A**pplications **M**eta-**M**odel) para un tipo de aplicaciones específico (por ejemplo prevención de desastres naturales o asistencia domiciliaria).

Este capítulo se centra fundamentalmente en la aproximación de modelado CADAMM. Así, el primer apartado está dedicado a analizar la problemática global del tipo de aplicaciones de interés. Es decir, aplicaciones de supervisión que se adaptan al

contexto Para ellos se analizan tres ámbitos diferentes, identificando los requisitos generales demandados. Dicha aproximación de modelado se organiza en **vistas de dominio** que proporcionan a cada usuario una visión diferente del mismo sistema real, de manera que cada uno realice sus tareas atendiendo únicamente a aquellos *aspectos relevantes para* su *área de trabajo*. Por ello, en los siguientes apartados se describe el meta-modelo CADAMM desde los diferentes puntos de vista correspondientes a los diferentes participantes de la especificación y diseño de las aplicaciones: expertos y desarrolladores. Por cada vista se describe su léxico y sintaxis, así como las reglas de composición que deben cumplir los modelos correctos.

3.2 Identificación de Requisitos de las Aplicaciones

En este apartado se identifican los requisitos de las aplicaciones distribuidas sensibles al contexto, analizando tres dominios distintos: asistencia domiciliaria, alerta temprana de catástrofes medio-ambientales y video-vigilancia.

Todas ellas son aplicaciones distribuidas en entornos heterogéneos, ya que capturan información de su contexto, mediante diferentes tipos de sensores, para posteriormente procesarla por equipos con muy diferentes capacidades y en distintas localizaciones (***Requisito R1 – Distribución de aplicaciones***). En el caso de la asistencia domiciliaria generalmente se trata de sensores que monitorizan las constantes vitales de los residentes y/o sensores que capturan variables ambientales, mientras que en video-vigilancia los sensores predominantes son cámaras que captan la información del contexto. Del mismo modo, en aplicaciones de alerta temprana, los datos medio-ambientales se registran por medio de sensores inalámbricos, distribuidos por varias regiones.

El procesamiento de estas medidas permite una monitorización continua del contexto, siendo posible prever situaciones de riesgo y proporcionar la asistencia más adecuada en caso de emergencia. Además, tanto la toma de medidas como su procesamiento, a pesar de ser similares en diferentes situaciones, deben ser personalizados de acuerdo a la problemática de cada caso (***Requisito R2 –***

Adquisición, procesamiento y actuación personalizados). Por ejemplo, el pulso de un paciente se mide de forma similar aunque con objetivos muy diferentes, bien para detectar emergencias cardiacas o bien para comprobar la relajación del paciente, así como con requisitos temporales distintos. Incluso los umbrales para decidir si un valor de pulso es anormal varían de un paciente a otro. En el caso de incendios, no es lo mismo detectarlo mediante el análisis de la concentración de gases capturada con sensores, que hacerlo mediante observación aérea a través de cámaras. Incluso un sistema de detección de incendios para interior de edificios no es válido para zonas extensas y despobladas, ya que no considera, por ejemplo, las condiciones ambientales.

Además, estos sistemas se consideran flexibles y dinámicos desde el punto de vista de su escalabilidad (***Requisito R3 – Escalabilidad***). Por ejemplo, el número de pacientes de una residencia puede variar (nuevos pacientes o bajas de residentes), o puede que se necesite controlar nuevas regiones de un monte, sin que ello deba influir en el resto de aplicaciones en ejecución. Incluso la monitorización de un paciente podría variar debido a la evolución de su estado de salud, como ocurre con la monitorización de la glucosa en caso de diabetes gestacional, que suele desaparecer tras el parto. Por otro lado, los recursos disponibles en la infraestructura también pueden cambiar debido a altas o bajas de equipamientos.

Con respecto a la QoS, como ya se ha indicado en el capítulo 2, hay calidades comunes a todas las aplicaciones, tal es el caso de la frecuencia de adquisición de medidas. Sin embargo, en otras ocasiones la QoS se refiere a diferentes características específicas de aplicaciones concretas (***Requisito R4 – Calidad de servicio específica de aplicación***). Por ejemplo, en aplicaciones de detección temprana de incendios en áreas remotas y despobladas con el objetivo de maximizar el tiempo de vida de los sensores, únicamente un subconjunto de ellos está activo en cada momento, mientras que el resto se encuentra en modo de espera. Así, la QoS se refiere a la configuración de las diferentes regiones del área bajo supervisión, que está definida por tres parámetros: su configuración temporal – frecuencia de adquisición, T - su configuración numérica – número de sensores activos, N – y su configuración espacial

– concentración de sensores activos, S. De forma similar, la QoS en aplicaciones multimedia puede estar formada por diferentes parámetros tales como la velocidad del video (número de imágenes por segundo, en *Frames Per Second* (FPS)), su resolución (alto x ancho de la imagen en número de pixeles) o su modo de codificación (en tasa de bits, constante o variable) (Bellavista et al., 2003; Chalmers & Sloman, 1999).

Por lo general, una misma aplicación podrá ejecutarse en diferentes niveles de QoS, de manera que en cada nivel los parámetros toman diferentes valores discretos. En el ejemplo de alerta temprana, la configuración (T, N, S) de cada región puede ser variable en función del nivel de peligrosidad detectado: cuanto más riesgo, más sensores activos, más concentrados y mayor frecuencia de adquisición. Habitualmente, se distinguen tres niveles de riesgo, de menor a mayor: ordinario, alerta y alarma (Intrieri et al., 2012; Ramesh, 2014). En este caso, es la propia aplicación la que decide el cambio de nivel de QoS en función del nivel de peligrosidad detectado, como resultado de su ejecución (adaptabilidad).

Por otro lado, pueden existir aplicaciones que soporten cierta degradación de su calidad de servicio, pudiendo ejecutarse en niveles de calidad inferiores al deseado, y requiriendo en ese caso menos recursos del sistema. En estas situaciones, la plataforma de gestión puede cambiar la QoS de la aplicación en función de los recursos disponibles y la demanda de recursos en un instante dado (***Requisito R5– Calidad de servicio flexible***). Por ejemplo, en una aplicación de video-vigilancia para control de acceso basado en reconocimiento facial, aunque lo deseable es trabajar con video de alta resolución, se obtienen resultados aceptables usando video de menos resolución. Por lo tanto, en este caso los niveles de calidad de servicio se extienden con otros aceptables en caso de necesidad. Además, es la plataforma de gestión la encargada de la gestión dinámica de la QoS, decidiendo los cambios de nivel en base a la disponibilidad de recursos del sistema en un instante concreto.

Se debe tener en cuenta, también, que en ciertas ocasiones cada medida se debe tomar a una frecuencia concreta, y que en otros casos se debe tomar en respuesta a

un evento concreto, existiendo diferentes tipos de activación (***Requisito R6 – Tipos de activación***). Por ejemplo, en asistencia domiciliaria para la detección de posibles infecciones, la temperatura corporal se puede medir cada 6 horas, mientras que para la identificación de paros cardiacos el pulso se debe supervisar cada 10 minutos. En video-vigilancia, la velocidad del video resulta fundamental para determinados procesamientos como el seguimiento de la trayectoria de un objeto en movimiento. Es más, puede haber acciones de monitorización de variables cuya medida se deriva del procesamiento de otras, todas ellas con características temporales diferentes. Este es el caso de la tensión arterial que se debe medir 4 veces al día. Tal y como se explica en Jobbágy et al. (2006) una buena medida de la tensión se obtiene cuando el paciente está relajado, lo cual se puede comprobar mediante su frecuencia cardiaca. Por tanto, cada 6 horas se debe monitorizar el ritmo cardiaco (con una frecuencia de adquisición de 30 segundos) y si el paciente se relaja, efectuar una única medida de tensión.

Cabe destacar que estas aplicaciones supervisan sistemas dinámicos que pueden evolucionar hacia situaciones peligrosas, como ocurre con el incremento del pulso de un paciente siempre que se produce un paro cardiaco, o la rotura de una presa que puede provocar inundaciones. Por lo tanto, además de la monitorización habitual y la detección de las posibles alarmas, sería interesante conocer cómo reaccionar frente a ellas. En los ejemplos anteriores, avisar al equipo médico o a los servicios de emergencia, es la forma más habitual de responder ante una alarma. Sin embargo, en ocasiones las aplicaciones deberían evolucionar para poder hacer frente a cambios importantes en su contexto (***Requisito R7 – Adaptabilidad***). Por ejemplo, tras la detección de un incendio potencial podría ser interesante aumentar la frecuencia de adquisición y el número de sensores activos en una región del monte. En otros casos, se debería iniciar el procesamiento de nuevas variables, como al activar aplicaciones de reconocimiento facial tras la detección de fuga de un preso. Y otras veces podría ser necesario detener actividades de supervisión en curso, como en el anterior ejemplo de medida de tensión arterial, en el que al relajarse el paciente ya no es necesario controlar su pulso.

Por último, teniendo en cuenta la importancia de detectar situaciones potencialmente peligrosas y reaccionar tan pronto como sea posible, así como la información sensible que se maneja (datos de pacientes, información de seguridad de un banco o aeropuerto, etc.), resulta fundamental que las aplicaciones sean confiables. Por un lado, se debe garantizar su disponibilidad (***Requisito R8 – Disponibilidad***), de manera que en caso de interrupción esporádica del servicio debería restaurarse tan pronto como sea posible y de forma que no afecte a la aplicación. Conviene destacar el caso de aplicaciones dinámicas cuyo estado en un instante depende del estado en instantes previos. Por ejemplo, para poder determinar la trayectoria de un objeto en movimiento en señales de video, se deben conocer los resultados de procesamientos previos. Si el nodo en el que se realiza dicho procesamiento cae, el proceso de recuperación debe incluir también la recuperación de los últimos resultados. Por otro lado, se debe asegurar el acceso controlado a datos sensibles, garantizando la integridad y privacidad de dichos datos (***Requisito R9 – Seguridad***).

La Tabla 3-1 resume los requisitos identificados.

Tabla 3-1: Requisitos de aplicaciones distribuidas sensibles al contexto

REQUISITO	DESCRIPCIÓN
R1 Distribución de aplicaciones	Integración de sensores y equipos de procesamiento distribuidos sobre plataformas heterogéneas.
R2 Adquisición, procesamiento y actuación personalizados	Adecuación de las tareas de adquisición, procesamiento y actuación a las particularidades del campo de aplicación y del entorno bajo monitorización concreto: soporte de diferentes sensores, umbrales, etc.
R3 Escalabilidad	Número y tamaño de aplicaciones variable y manejable. Soporte a la modificación (inclusión/eliminación) de aplicaciones y recursos compartidos.
R4 Calidad de servicio específica de aplicación	Soporte a la especificación y aseguramiento de la calidad de servicio demandada por cada tipo de aplicación.

REQUISITO	DESCRIPCIÓN
R5 Calidad de servicio flexible	Capacidad para especificar y gestionar los distintos niveles de demanda de recursos soportados por una aplicación.
R6 Tipos de activación	Tareas de medida o procesamiento activadas por tiempo o bajo demanda.
R7 Adaptabilidad	Sensibilidad a cambios en el contexto: lanzamiento/parada de aplicaciones, modificación de sus propiedades…
R8 Disponibilidad	Asegurar la continuidad del servicio, o la restauración tan pronto como sea posible sin afectar a las aplicaciones.
R9 Seguridad	Asegurar la privacidad, confidencialidad e integridad de los datos gestionados por las aplicaciones.

A lo largo de los siguientes apartados se presentan los mecanismos de modelado ofrecidos a los diferentes expertos que participan en la especificación y diseño de las aplicaciones para capturar el diseño de las aplicaciones, cumpliendo con los requisitos identificados, así como los requisitos que debe cumplir la plataforma de gestión. Dichos mecanismos se agrupan en vistas de expertos.

3.3 Vista del Experto de Dominio: Diseño Arquitectónico

En este apartado se describen los mecanismos de modelado para que un experto pueda especificar aplicaciones de su dominio, cumpliendo parte de los requisitos identificados y determinando así su **diseño arquitectónico**. Los conceptos de modelado propuestos, las relaciones entre ellos y el conjunto de reglas de composición conforman la Vista del Experto de Dominio del meta-modelo CADAMM, que se representa gráficamente por medio de diagramas de clase UML (Booch et al., 2005).

Para cumplir el requisito de **distribución (R1)** se propone el concepto de **Componente** como unidad de ejecución indivisible que representa una tarea de adquisición de datos, procesamiento o actuación sobre el entorno, y que posteriormente se implementará como un componente, servicio o agente.

Cada actividad de supervisión personalizada (p. ej., monitorización del ritmo cardiaco en un paciente con problemas de corazón, o el control de acceso a una sala en base a reconocimiento facial) se corresponde con el concepto de **Aplicación Atómica** que está formada por un conjunto de componentes de adquisición, procesamiento y/o actuación que cooperan para alcanzar el objetivo funcional de su aplicación, y cuyas instancias pueden ejecutarse en nodos heterogéneos y distribuidos, demandando recursos de la infraestructura. En efecto, estos componentes que especifica el experto de dominio se implementan en forma de *componentes software* en fases sucesivas del ciclo de desarrollo. A modo de ejemplo, una aplicación atómica para el control de la temperatura corporal de un paciente se puede definir en base a 4 componentes: (1) adquisición de la temperatura; (2) almacenamiento en el historial del paciente; (3) comprobación de si el valor capturado está fuera del rango normal del paciente; y (4) aviso al personal médico en caso de una situación anormal. La cooperación entre componentes se define a través de la **Lógica de Interconexión de Datos** que expresa los datos a intercambiar, cuándo y entre qué componentes (**R2 – Personalización**).

Con respecto a la **Escalabilidad (R3)**, el concepto de **Escenario** permite agrupar un conjunto de aplicaciones que tienen algo en común, de manera que el experto puede hacer crecer el sistema de una forma organizada y sencilla, bien extendiendo las aplicaciones de un escenario o definiendo nuevos escenarios. Es importante destacar que el concepto de escenario puede tener significados muy diferentes en cada ámbito de aplicación. Por ejemplo, en asistencia domiciliaria puede referirse a un paciente, englobando todas sus actividades de monitorización, mientras que en aplicaciones de alerta temprana de desastres un escenario puede corresponderse con cada una de las regiones en las que se divide la zona bajo supervisión. En cuanto a la escalabilidad de los recursos compartidos por las aplicaciones (altas o bajas de equipamientos de

medición, actuación o procesamiento), cabe destacar que no son objeto del diseño de las aplicaciones, siendo responsabilidad de la plataforma de gestión.

A la hora de detallar la **calidad de servicio específica de aplicación (R4)**, el principal reto reside en encontrar una forma genérica de recoger las diferentes características que la pueden conformar, de manera que puedan definirse nuevas QoS de forma fácil y rápida. Se propone como solución recoger dicha información como un elemento de modelado parametrizado, tomando como referencia los lenguajes QML (Frølund & Koistinen, 1999) y QDL (Loyall et al., 1998). Más concretamente, una QoS se corresponde con un **Grupo de QoS** que agrupa un conjunto de **Tipos de QoS**, cada uno de los cuales se relaciona con una de las características que definen la QoS. Así, la configuración de las regiones de una zona montañosa monitorizada podría definirse como un grupo de QoS formado por tres tipos: su configuración temporal (T), su configuración numérica (N) y su configuración espacial (S). Por su parte, el concepto de **Nivel de QoS** permite identificar los diferentes niveles de calidad que se corresponden con diferentes niveles de demanda de recursos que puede soportar una aplicación (**R5 – Calidad de Servicio flexible**) y que se refieren a los mismos tipos de QoS pero con diferentes valores. El concepto de Nivel de QoS también podría representar un cambio en la calidad de servicio como ser parte de la reacción ante un cambio de contexto (**R7 – Adaptabilidad**).

Además, el concepto de Grupo de QoS se puede asociar tanto a un componente para definir, por ejemplo, la calidad del video que procesa, como a una aplicación para especificar, por ejemplo, su periodo de activación. De hecho, el requisito temporal periodo es en realidad un tipo particular de QoS aplicable a componentes o aplicaciones *activadas por tiempo*. El concepto de **Súper-Aplicación** permite definir actividades de supervisión más complejas agrupando aplicaciones atómicas, de manera que los requisitos temporales de la súper-aplicación condicionan la activación de todas sus aplicaciones atómicas en conjunto. La *activación bajo demanda* de los componentes queda especificada por la lógica de interconexión de datos, mientras que la de las aplicaciones está directamente relacionada con el

requisito de adaptabilidad, cubriéndose así diferentes **tipos de activación (R6)**, temporizados y no temporizados.

Precisamente, el concepto de **Evento** permite identificar el suceso de un cambio de contexto relevante frente al que hay que reaccionar **(R7 – Adaptabilidad)**. En última instancia, los responsables de la detección de eventos son los componentes por lo que se les proporciona un **Puerto de Eventos** donde el experto de dominio puede definir la **Lógica de Lanzamiento de Eventos**. Siguiendo la filosofía del paradigma *Event-Condition-Action* (Sadri, 2011; Ballagny et al., 2009) descrita en el capítulo del estado del arte, la respuesta a un evento se determina por medio de un conjunto de **Acciones** que se deben llevar a cabo, sin intervención humana, para arrancar nuevas aplicaciones, detener la ejecución de aplicaciones existentes o modificar el nivel de QoS en el que se deben ejecutar.

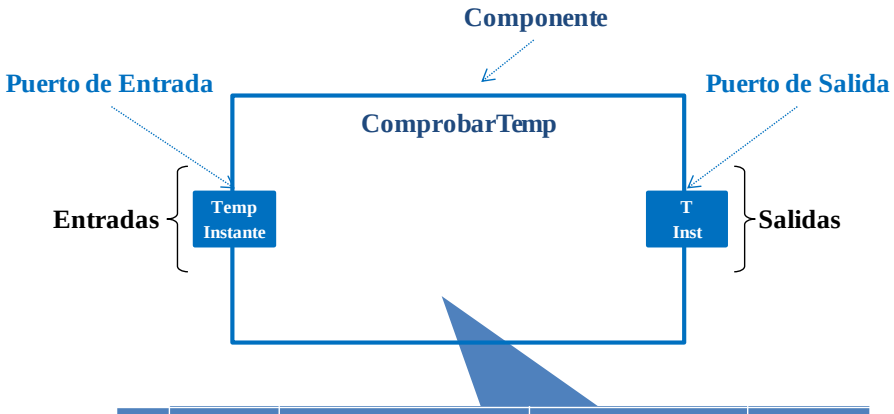
En los siguientes sub-apartados se detalla cada elemento de modelado, el papel que juega en la vista y sus principales características.

3.3.1 Componente

Los componentes encapsulan una funcionalidad que puede precisar ciertos datos de entrada y que puede proporcionar determinados datos de salida, de manera que el experto de dominio debe identificar estos **Datos** que maneja el componente, indicando para cada uno: *nombre* del dato (p. ej., pulso o concentración de CO_2), una *descripción* y las *unidades* (pulsaciones por minuto o % en los ejemplos anteriores). A modo de ejemplo, la Figura 3-2 muestra el detalle de uno de los componentes de la aplicación *TempCorporal* que monitoriza la temperatura corporal. Más concretamente, se trata del componente *ComprobarTemp* que verifica si la medida realizada se encuentra dentro del rango considerado normal para un paciente. Los datos de entrada y salida de dicho componente se recogen en la tabla de la Figura 3-2. En concreto, necesita recibir dos de entrada: 1) *Temp*, temperatura medida en ºC; 2) *Instante*, momento de captura de la medida en formato "dd-mm-aaaa hh:mm:ss". Y proporciona tres de salida: 1) *Temp* e 2) *Inst*, similares a los de entrada, y 3) *enRango*

que indica si la medida se considera normal para el paciente, tomando los valores verdadero o falso. Nótese que para realizar su función es posible que consulte la base de datos del paciente con el objetivo de conocer su rango de valores normales.

Los componentes de una aplicación atómica cooperan intercambiándose información para proporcionar la funcionalidad requerida. Por lo tanto, los componentes no sólo encapsulan una funcionalidad sino también la lógica de conexión con aquéllos que les suministran los datos de entrada (si los precisa) así como con aquéllos a los que suministra sus datos de salida (en caso de que los tenga). Para ello, como se observa en la Figura 3-2, están provistos de un **Puerto de Entrada** encargado de recibir datos de sus predecesores (**Entradas**) y/o un **Puerto de Salida** que recoge los datos resultado de su funcionalidad para enviarlos a sus siguientes (**Salidas**).



DATOS ENTRADA/SALIDA

Nombre	Descripción	Unidades	Dirección
Temp	Temperatura	ºC	Entrada
Instante	Instante de medida	dd-mm-aaaa hh:mm:ss	Entrada
T	Temperatura	ºC	Salida
Inst	Instante de medida	dd-mm-aaaa hh:mm:ss	Salida
enRango	¿Dentro del rango normal?	V / F	Salida

Figura 3-2: Detalle del componente *ComprobarTemp*.

3.3.2 Lógica de Interconexión de Datos

La interconexión entre dos componentes se realiza a través de sus puertos de entrada y salida, enlazados por un **Conector de Datos** que agrupa la información

intercambiada. Este intercambio de datos se representa por medio de **Conexiones de Datos** que relacionan cada entrada de un componente con una salida de un predecesor.

En base a todos estos conceptos, la cooperación entre componentes se puede representar por medio de un grafo similar al de la Figura 3-3, que se corresponde con la aplicación atómica *TempCorporal* antes descrita. Por simplicidad, se han omitido las tablas que caracterizan los datos de cada componente. Los conectores de datos se representan mediante flechas negras, mostrándose el detalle de las conexiones de datos para cada uno de ellos. Nótese que las salidas enviadas por el componente *Adquisición* al componente *Almacenamiento* no son iguales a las enviadas al componente *ComprobarTemp*. De hecho, el establecer conexiones de datos particulares de cada conector, permite seleccionar qué salidas, de todas las disponibles, se envían a cada componente siguiente.

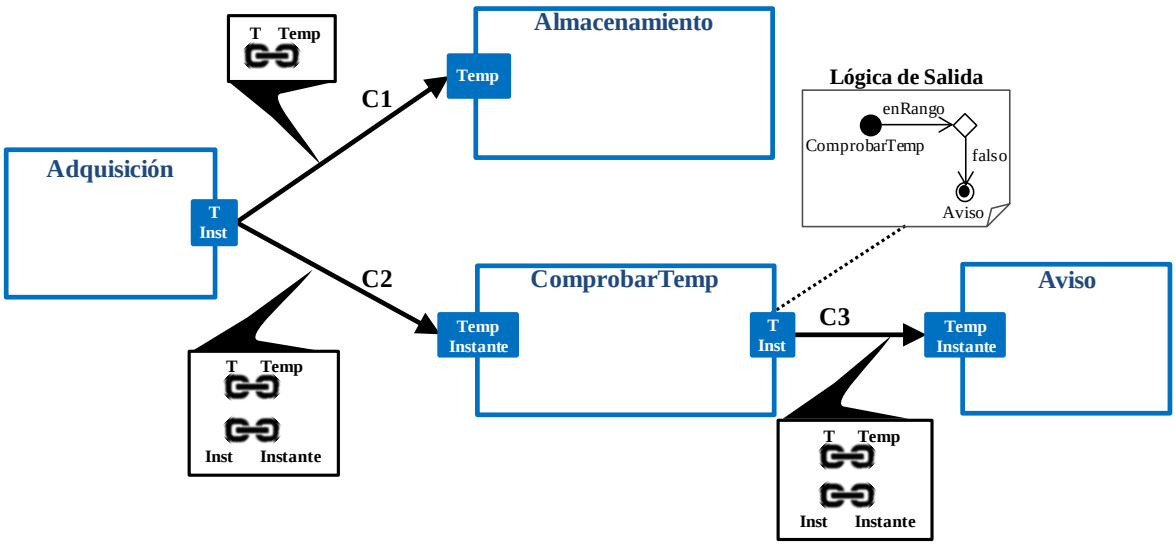


Figura 3-3: Grafo de componentes de la aplicación atómica *TempCorporal*

Por defecto, se asume que las salidas del componente siempre se envían a todos los siguientes según las conexiones de datos especificadas en los conectores de datos correspondientes. Este es el caso del componente *Adquisición* de la Figura 3-3 que siempre envía la temperatura al componente *Almacenamiento* para guardarla en el

historial del paciente, y siempre envía la temperatura y el instante de su medida al componente *ComprobarTemp* para analizarlos.

Sin embargo, en ocasiones es necesario *personalizar la lógica del envío de las salidas*. Esta lógica se define mediante el concepto **Lógica de Salida** que se representa por medio de un diagrama de actividades UML (Booch et al., 2005) asociado al puerto de salida. En la aplicación de la Figura 3-3 se presenta un ejemplo de este diagrama de actividades asociado al puerto de salida del componente *ComprobarTemp*, donde se determina que únicamente se avisará al personal médico (envío de salidas al componente *Aviso*) si la temperatura está fuera del rango normal del paciente (dato de salida *enRango* tiene valor falso). El Nodo Inicial de este diagrama se corresponde con el componente actual (componente *ComprobarTemp*) mientras que cada Nodo Final hace referencia a uno de sus siguientes (sólo uno para el componente *Aviso*). Los Flujos de Control se basan en expresiones que contienen datos de salida del componente. Por simplicidad, únicamente se consideran expresiones con combinaciones lógicas de dichos datos, aquéllas cuyo resultado es verdadero o falso. Al igual que ocurre con el componente *Adquisición*, tampoco todos los datos de salida del componente *ComprobarTemp* se relacionan con salidas que se envían al siguiente componente. Existen también datos de salida que se pueden usar como datos de decisión (p. ej., dato de salida *enRango*).

De forma similar, también se puede establecer una **Lógica de Entrada** que define la obtención de los datos de entrada del componente a partir de las entradas recibidas, a través de conectores de datos, en su puerto de entrada. De nuevo, esta lógica se expresa mediante un diagrama de actividades UML asociado a dicho puerto. En este caso, el Nodo Inicial representa al puerto de entrada, mientras que hay tantos Nodos de Salida como conectores de datos llegan al puerto de entrada. Los Flujos de Control se basan en expresiones que contienen entradas recibidas en el puerto de entrada.

3.3.3 Aplicación: Súper-Aplicación y Aplicación Atómica

Como se ha comentado anteriormente, una **Aplicación Atómica** representa una actividad de supervisión personalizada formada por componentes que interactúan intercambiando datos. Se trata de un elemento que permite al experto de dominio identificar qué variables hay que supervisar y cómo procesarlas.

Por su parte, la **Súper-Aplicación** tiene la responsabilidad de lanzar de forma temporizada las aplicaciones que contiene. Se trata de aplicaciones atómicas que tienen requisitos temporales relacionados de alguna forma, por ejemplo que todas deben activarse a la vez, por lo que este elemento también hace más sencilla la modificación (extensión/disminución) de aplicaciones concretas.

3.3.4 Escenario

El concepto de **Escenario** permite agrupar un conjunto de aplicaciones (súper-aplicaciones y/o aplicaciones atómicas) que tienen algo en común. En la Figura 3-4 se ilustra este concepto mediante la especificación de una residencia (**Sistema**) con tres pacientes (Escenarios) y la supervisión de las zonas comunes (Escenario). En dicha figura se observa que a todos los pacientes se les ha asignado la aplicación atómica llamada *MonitorizaciónEmergencia* para la monitorización excepcional de constantes vitales básicas en caso de incendio, con el objetivo de facilitar y agilizar la atención de los servicios de emergencia. Además, al *Paciente1* se le supervisa la temperatura corporal, ya que ha sido operado y no presenta ningún otro problema de salud relevante (aplicación atómica *TempCorporal*). Por su parte, el *Paciente2* padece hipertensión por lo que su tensión arterial se debe controlar con una cierta frecuencia (súper-aplicación *TensiónArterial*). Se toma la medida de tensión (aplicación atómica *MedirTensión*), comprobando que previamente se ha relajado (aplicación atómica *ComprobarRelajación*). Por último, el *Paciente3* se corresponde con un residente con problemas de corazón, demandando el control de su ritmo cardiaco (aplicación atómica *ControlPulso*).

Para poder definir todos los escenarios y aplicaciones necesarios, el experto de dominio debe conocer el entorno físico (representado por *M_Entorno* en la Figura 2-1).

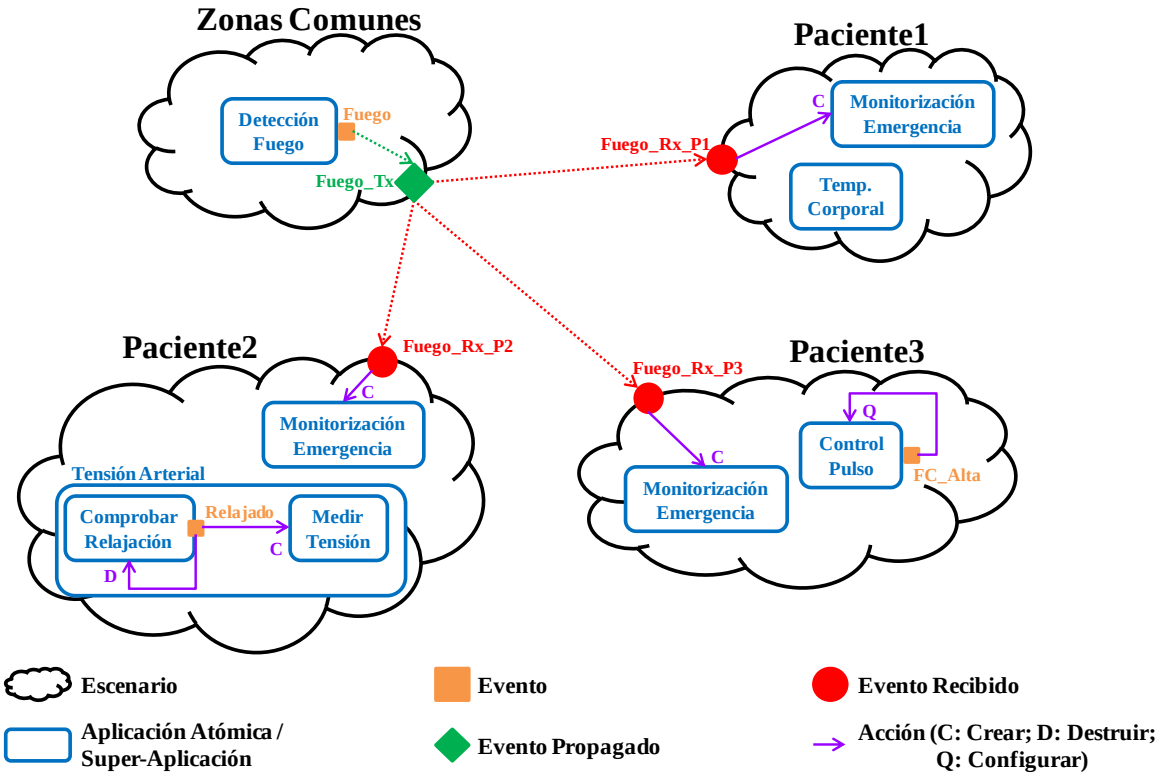


Figura 3-4: Representación gráfica de la especificación de una residencia con tres pacientes.

Esta estructura del *Diseño Arquitectónico* permite al experto de dominio hacer crecer al sistema de una forma sencilla. Por ejemplo, en un sistema de alerta temprana puede ser que aumente la zona bajo supervisión, lo que implica un incremento de las regiones del sistema (escenarios). Algo similar ocurre si en un aeropuerto con sistema de video-vigilancia se habilita un nuevo acceso (escenario). En ambos casos se producen cambios en el número de escenarios en el sistema. Incluso dentro de un escenario también puede haber variaciones en sus aplicaciones atómicas o súper-aplicaciones. Por ejemplo, si el estado de salud de un paciente varía con el tiempo, sería necesario alterar el diseño previo de su escenario, añadiendo nuevas aplicaciones (el paciente presenta problemas de corazón que nunca había padecido), o eliminando aplicaciones ya definidas (el periodo de post-operatorio ha finalizado y ya no es necesario supervisar la temperatura corporal).

3.3.5 Grupo y Tipo de QoS

Existen QoS determinadas por una única característica como el caso de la frecuencia de adquisición de datos de un sensor. Pero también hay otras más complejas como la configuración (T, N, S) de una región montañosa monitorizada, o la calidad del video que se procesa en aplicaciones multimedia, definida por los FPS y su resolución. Cada una de estas características se corresponde con un **Tipo de QoS**, que pueden ser agrupados en **Grupos de QoS**. Cada tipo de QoS está caracterizado por su *nombre*, las *unidades* en las que se mide, y el *valor* que se le asigna.

Las aplicaciones de interés monitorizan su contexto mediante dispositivos de medición (como por ejemplo sensores de concentración de CO_2, sensores de temperatura, pulsioxímetros, glucómetros, o cámaras de video), teniendo en cuenta que cada medida se debe realizar a una frecuencia determinada. Por ejemplo, en el caso del *Paciente1* (ver Figura 3-4) la temperatura se mide cada 6 horas, mientras que el pulso del *Paciente3* se toma cada 10 minutos. Sin embargo, no todas las aplicaciones sensibles al contexto son periódicas. Por ejemplo, la aplicación *MedirTensión* del *Paciente2* toma una única medida de tensión arterial, siendo la relajación del paciente la situación que provoca su activación. En resumen, se distinguen dos tipos de activación de aplicaciones: (1) *por tiempo*, el periodo de la aplicación determina el instante de activación; (2) *bajo demanda*, cuando es la detección de una situación concreta el detonante de su activación. Este último está directamente relacionado con el requisito de adaptabilidad que se detalla en el apartado 3.3.7.

En caso de activación por tiempo, el experto de dominio debe indicar el periodo de la aplicación (atómica o súper-aplicación), que será un tipo de QoS perteneciente a un grupo de QoS para los requisitos temporales. Para la aplicación atómica *TempCorporal* el periodo se mide en horas con un valor de 6, mientras que para la aplicación atómica *ControlPulso* se mide en minutos con un valor de 10. Mención especial merece el caso de las súper-aplicaciones cuyos requisitos de activación afectan al grupo de sus atómicas. En el caso de la súper-aplicación *TensiónArterial*, el

tipo de QoS indica que su periodo se mide en horas y con un valor de 4. Su aplicación atómica *ComprobarRelajación* tiene un periodo medido en segundos con valor de 30, mientras que la aplicación atómica *MedirTensión* es activada bajo demanda, por lo que no lleva asociado dicho grupo de QoS para requisitos temporales. Esta definición de QoS implica que cada 4 horas se debe activar la medición de la tensión arterial del paciente, lo que supone el arranque de la aplicación *ComprobarRelajación* en primer lugar (que se ejecuta según su periodo) y el de la aplicación *MedirTensión* cuando el paciente se relaje (ver apartado 3.3.7).

Por otro lado, los componentes pueden ser *periódicos* (su periodo determina el instante de activación), *de ejecución única* (el instante de activación coincide con el de su aplicación) o *esporádicos* (la activación se produce tras la recepción de sus datos de entrada, en función de la lógica definida). Los dos primeros casos se refieren a componentes iniciales de una aplicación. Nótese que en una aplicación puede haber varios componentes iniciales.

3.3.6 Nivel de QoS

Como se ha comentado, este concepto permite definir dos situaciones. Por un lado, los diferentes *niveles de calidad en los que una aplicación debe ejecutarse*, como es el caso de las diferentes configuraciones de una zona montañosa en función del nivel de peligrosidad (p. ej., riesgo de incendio). Por otro lado, los diferentes *niveles de calidad en los que una aplicación puede ejecutarse* ya que soporta cierta degradación de su QoS (QoS flexible), como ocurre con determinadas aplicaciones multimedia.

En ambas situaciones, el experto debe identificar cuáles son los **posibles niveles**, teniendo en cuenta que todos ellos comparten los mismos tipos de QoS, pero con diferente valor en cada nivel. Así, en una aplicación de video-vigilancia podrían identificarse dos niveles de calidad, alta y baja, con diferentes valores de velocidad de video (por ejemplo, 25FPS y 8FPS, respectivamente) y resolución (por ejemplo, 1280x720 pixeles y 352x240 pixeles, respectivamente).

El primer caso entra dentro de las necesidades de adaptabilidad de la aplicación por lo que se detalla en el siguiente apartado. En el caso de **QoS flexible**, dentro del grupo de QoS los *niveles* deben estar *ordenados* por su grado de degradación de calidad. En última instancia, esta flexibilidad de QoS se traduce en *demanda de recursos variable* en función del nivel, a mayor nivel mayor demanda. En el caso de aplicaciones multimedia, la demanda de recursos de aquellos componentes que procesen video o imágenes variará en función, por ejemplo, del tamaño de la imagen a procesar. En cambio, aquéllos que no las procesen mantendrán su demanda constante, a pesar de cambios de calidad. Por tanto, se deben identificar también los **componentes** que se ven **afectados** por dicha flexibilidad. Teniendo en cuenta que pueden ser varios los componentes afectados, por simplicidad se propone definir un único grupo de QoS a nivel de aplicación, relacionándolo con los componentes afectados.

Por último, se debe indicar cuál o cuáles son los **componentes que realizan el cambio de nivel**. Por ejemplo, en el caso de aplicaciones de video suele ser un único componente, el que gestiona la cámara, ya que modificando la configuración de dicha cámara se modifica la calidad del video y, por lo tanto, la demanda de recursos de todos los componentes que lo procesen.

3.3.7 Condición – Evento – Acción

Para definir la adaptación al contexto es necesario que el experto pueda especificar qué cambios de contexto son relevantes, cómo identificarlos y cómo reaccionar frente a ellos.

En este sentido, el concepto de **Evento** permite identificar el suceso de un cambio de contexto relevante en una aplicación atómica. Por ejemplo, en la monitorización de la frecuencia cardiaca para el *Paciente3*, un aumento anormal del pulso se considera un cambio de contexto relevante, representado por el evento *FC_Alta* en la Figura 3-4. En cambio, para el *Paciente2* resulta importante el instante en el que se relaja (evento *Relajado* en la Figura 3-4), lo cual se detecta mediante el procesamiento de varios valores de pulso consecutivos. Del mismo modo, en el caso de un centro penitenciario,

un evento puede representar un intento de fuga de un preso, identificado tras analizar si un objeto en movimiento ha cruzado una determinada línea virtual.

Estos ejemplos muestran que el contexto se corresponde con el entorno bajo supervisión y que la detección de cambios de contexto se basa en resultados del procesamiento de los componentes. Más concretamente, se puede expresar en base a determinados datos de salida. Es por ello que se dota a los componentes de un **Puerto de Eventos** que lleva asociado un diagrama de actividades donde se recoge la lógica definida por el experto de dominio para el lanzamiento de eventos (**Lógica del Evento**). En la Figura 3-5 se muestra el grafo de la aplicación *ComprobarRelajación* que, mediante la supervisión del pulso del *Paciente2*, comprueba si alcanza el estado de relajación suficiente para medirle la tensión arterial (evento *Relajado* en Figura 3-4). La lógica para el lanzamiento del evento se representa en el diagrama de actividades asociado al puerto de eventos del componente *ComprobarRelajado*. Como se puede observar, se trata de un diagrama similar al de la lógica de salida, con un Nodo Inicial que representa al componente actual, y Controles de Flujo basados en sus datos de salida. Sin embargo, en este caso cada Nodo Final representa el evento que se lanzará al detectarse el cambio de contexto. Por simplicidad, en el grafo de la Figura 3-5 se han omitido los nombres de los conectores de datos así como sus conexiones de datos.

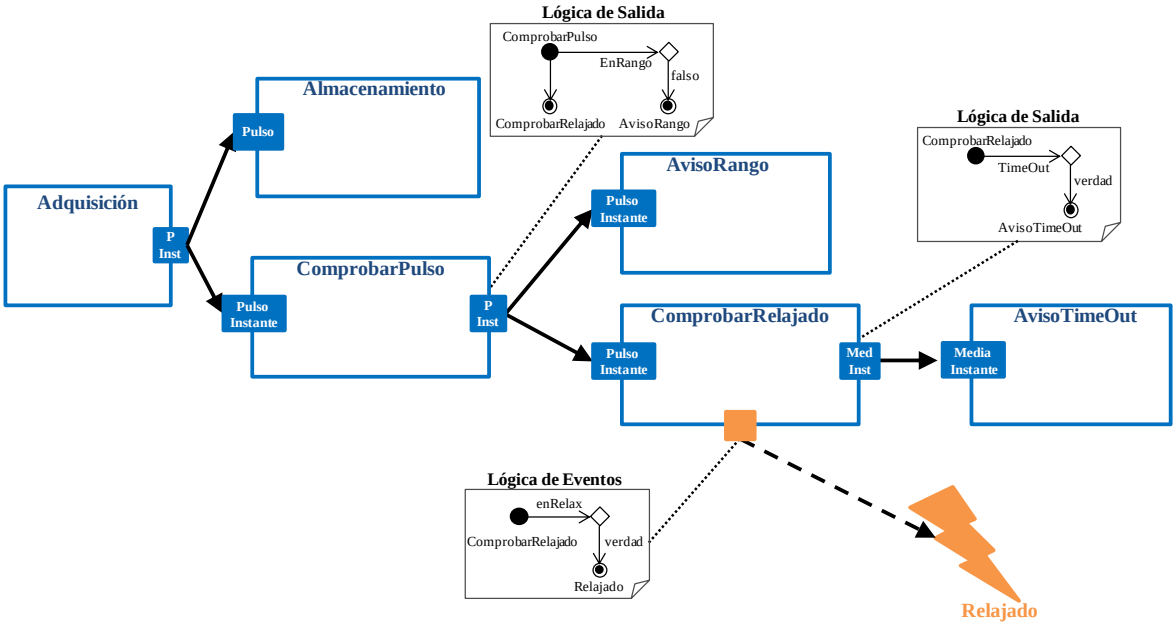


Figura 3-5: Grafo de componentes de la aplicación ComprobarRelajación

Con respecto a la reacción frente a un cambio de contexto, el simple aviso resultado de la lógica de salida, mostrado en las figuras Figura 3-3 y Figura 3-5, resulta insuficiente, ya que lo que se busca es que el comportamiento de la aplicación evolucione conforme a la evolución de su contexto. Por lo tanto, dicha reacción se plasma como un conjunto de **Acciones** que se deben ejecutar tras el lanzamiento del evento correspondiente, sin intervención humana, y siguiendo, si fuera necesario, un determinado *orden*. Es la plataforma de gestión la encargada de asegurar que dicho orden se respete en tiempo de ejecución. En la fase de diseño arquitectónico, el experto únicamente debe indicar qué acciones deben ejecutarse en secuencia y su orden. También puede establecer un *tiempo límite* para su atención. Es decir, el tiempo máximo del que dispone la plataforma de gestión para ejecutar todas las acciones asociadas a dicho evento. Se distinguen tres tipos de acciones: Crear, Destruir y Configurar. Todas ellas tienen como *objetivo* una aplicación atómica o súper-aplicación, que puede ser la misma que lanzó el evento u otra diferente.

Para ilustrar el uso de eventos, en el escenario *Paciente 2* de la Figura 3-4 se observa que tras el lanzamiento del evento *Relajado* se ejecutan dos acciones, una para detener la monitorización del pulso y otra para iniciar la medida de la tensión arterial. La primera se corresponde con una acción de tipo **Destruir** (acción *D* en la

Figura 3-4), mientras que la segunda con una de tipo **Crear** (acción *C* en la Figura 3-4). Además, la nueva aplicación se puede arrancar con un estado de ejecución inicial para determinados componentes, que se obtiene del estado de ejecución de componentes de la aplicación que lanza el evento. Así, siguiendo la filosofía de conexiones de datos, se definen conexiones de estados entre un componente de la aplicación destino y un componente de la aplicación actual. Tal es el caso de una aplicación que procesa imágenes para seguimiento de objetos en movimiento y, tras un evento, es sustituida por otra, que además del seguimiento, dispone de nuevas funcionalidades. Es necesario que el componente que analiza trayectorias de la nueva aplicación disponga de los datos de su homólogo, es decir, de su estado de ejecución.

Una acción de tipo **Configurar** permite modificar la QoS de una aplicación. En la monitorización de la frecuencia cardiaca del *Paciente3* (ver Figura 3-4), tras el lanzamiento del evento *FC_Alta* se ejecuta una acción de tipo *Configurar* (acción *Q* en la Figura 3-4) para aumentar la frecuencia de medida del pulso. En este caso, la aplicación objetivo de la acción es ella misma, siendo necesario indicar la nueva frecuencia de adquisición, representada por un Grupo de QoS de la aplicación. Otro ejemplo se puede encontrar en una aplicación de alerta temprana de desastres medio-ambientales en la que se identifican tres niveles de alerta (ordinario, alerta y alarma), a cada uno de los cuales le corresponde una configuración (T, N, S) diferente. En este caso, el experto definirá que cuando se produzca un cambio del nivel de riesgo de incendio se lance el evento correspondiente. Dicho evento debe tener asociada una acción de tipo *Configurar*, en la que se indica el grupo de QoS con los valores de tipos correspondientes al nuevo nivel de riesgo (nivel de QoS). Nótese, que en este caso de adaptabilidad es necesario que el experto de dominio también identifique cuál es el *nivel de QoS inicial* en el que se iniciará la ejecución de la aplicación por primera vez.

Por último, si el objetivo de una acción es una súper-aplicación, en caso de ser de tipo *Crear* no se indican estados de ejecución iniciales; si es de tipo *Configurar*, únicamente se podrán modificar los grupos de QoS relativos a la súper-aplicación, es decir, sus requisitos temporales. Finalmente, si se trata de una acción de tipo *Destruir*,

se detendrá la ejecución de todas sus atómicas y se eliminarán recuperando recursos del sistema.

3.3.8 Propagación de Eventos

En todos los ejemplos del apartado anterior, las acciones de un evento siempre han tenido como objetivo una aplicación del mismo escenario. Es decir, los eventos relacionados con un paciente sólo han repercutido en aplicaciones del mismo paciente. Sin embargo, en ocasiones las acciones derivadas de un cambio en el contexto deben sobrepasar las fronteras de su propio escenario, para lo cual se permite la propagación de eventos entre escenarios. Este es el caso de la detección de un incendio en la residencia (evento *Fuego* en Figura 3-4), que provoca la activación de una monitorización de emergencia en todos los residentes. Otro ejemplo se puede encontrar en una aplicación de alerta temprana que detecte una situación de alarma en una región (escenario) y la quiera propagar a otra región (otro escenario). Por lo tanto, un Escenario puede propagar eventos de sus aplicaciones a otros escenarios (**TxEventoEscn**), y también puede recibir eventos que han sido propagados por otros escenarios (**RxEventoEscn**), y que a su vez llevan asociadas acciones cuyo objetivo son aplicaciones del escenario receptor. Para ilustrar todos estos conceptos, en la Figura 3-4 se observa cómo el escenario *Entorno* propaga el evento *Fuego_Tx* (que proviene del evento *Fuego* de su aplicación *DetecciónFuego*) a todos los escenarios relativos a pacientes de la residencia (*Fuego_Rx_P1*, *Fuego_Rx_P2*, *Fuego_Rx_P3*). En cada escenario de paciente el evento recibido tiene asociada una acción de tipo *Crear* cuyo objetivo es la aplicación *MonitorizaciónEmergencia* correspondiente.

3.3.9 Meta-modelo y Reglas de Composición

Debido a la complejidad de las relaciones entre elementos, todos los conceptos de modelado identificados en esta vista así como las relaciones entre ellos se capturan en tres diagramas de clases. La Figura 3-6 se refiere a los elementos relacionados con los requisitos R1-R6 (i. e. distribución, personalización, escalabilidad, QoS específica de aplicación, tipos de activación y QoS flexible), mientras que la Figura 3-7 a los

relacionados con el requisito de adaptabilidad (R7). En esta última figura, los elementos nuevos se han resaltado en color amarillo. Nótese que los elementos empleados para la definición de la lógica de entrada, salida y lanzamiento de eventos son un subconjunto del diagrama de actividades definido por UML (ver Figura 3-8).

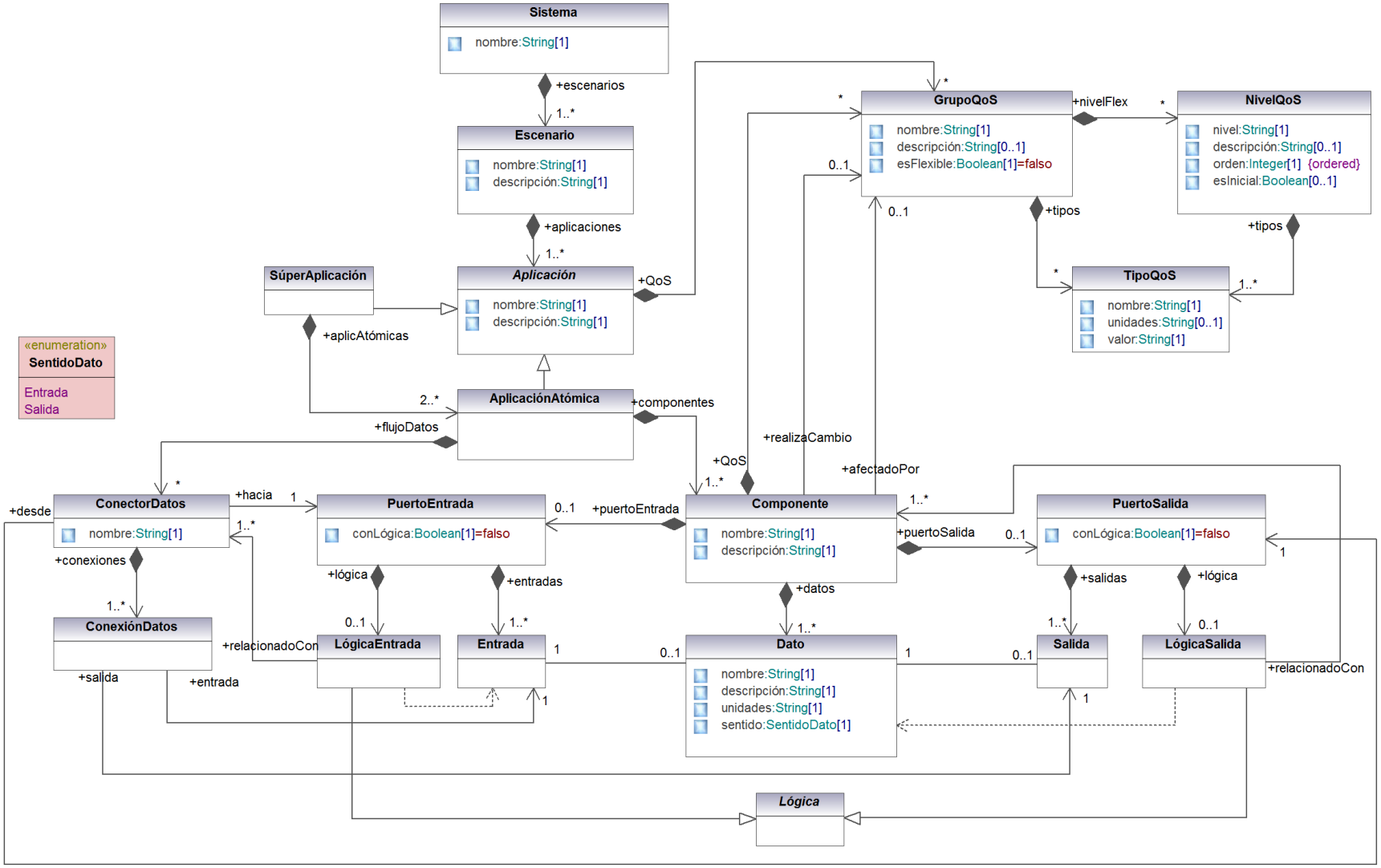


Figura 3-6: Detalle de la Vista del Experto de Dominio del meta-modelo CADAMM: elementos relacionados con los requisitos R1-R6

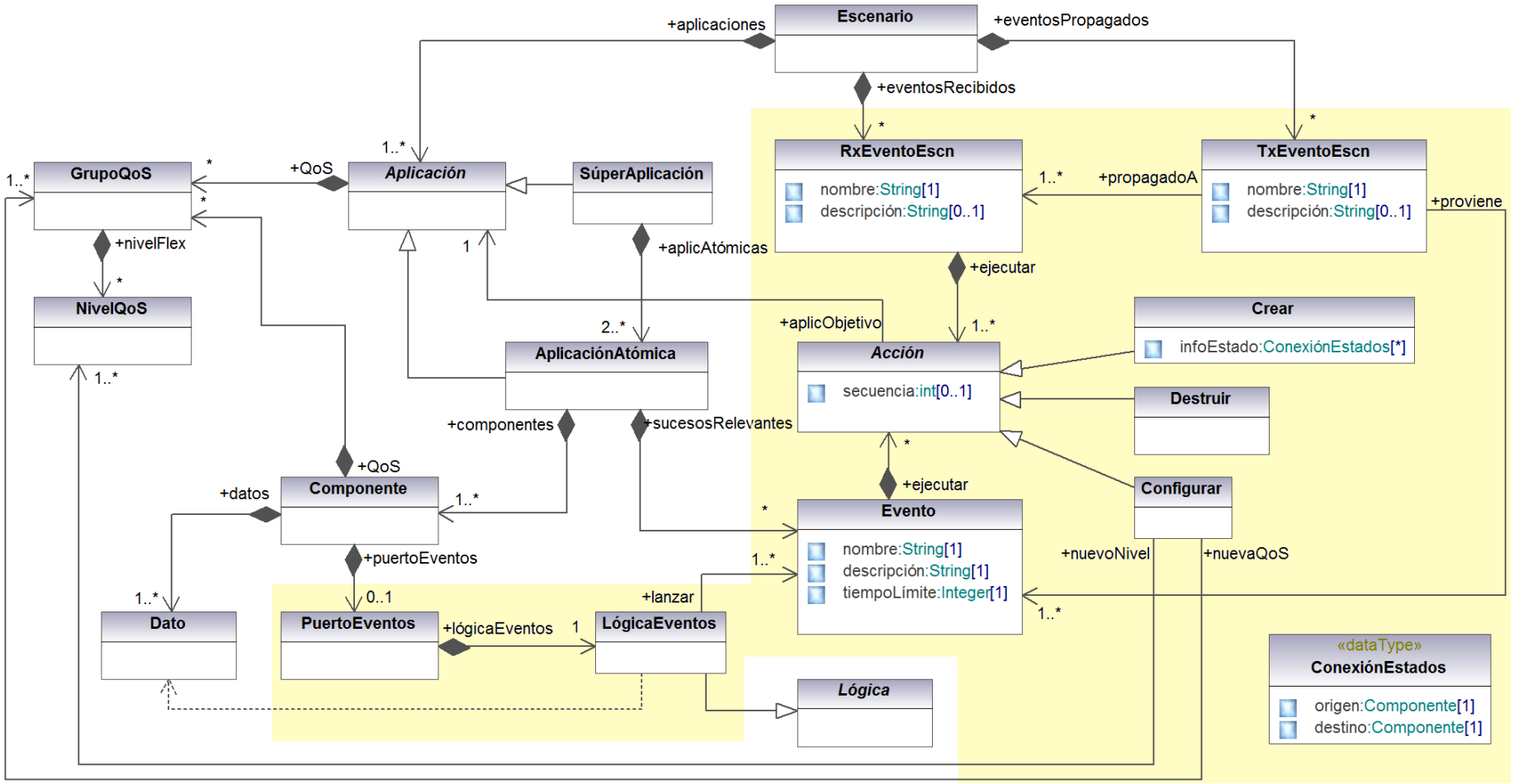


Figura 3-7: Detalle de la Vista del Experto de Dominio del meta-modelo CADAMM: elementos relacionados con el requisito R7: Adaptabilidad

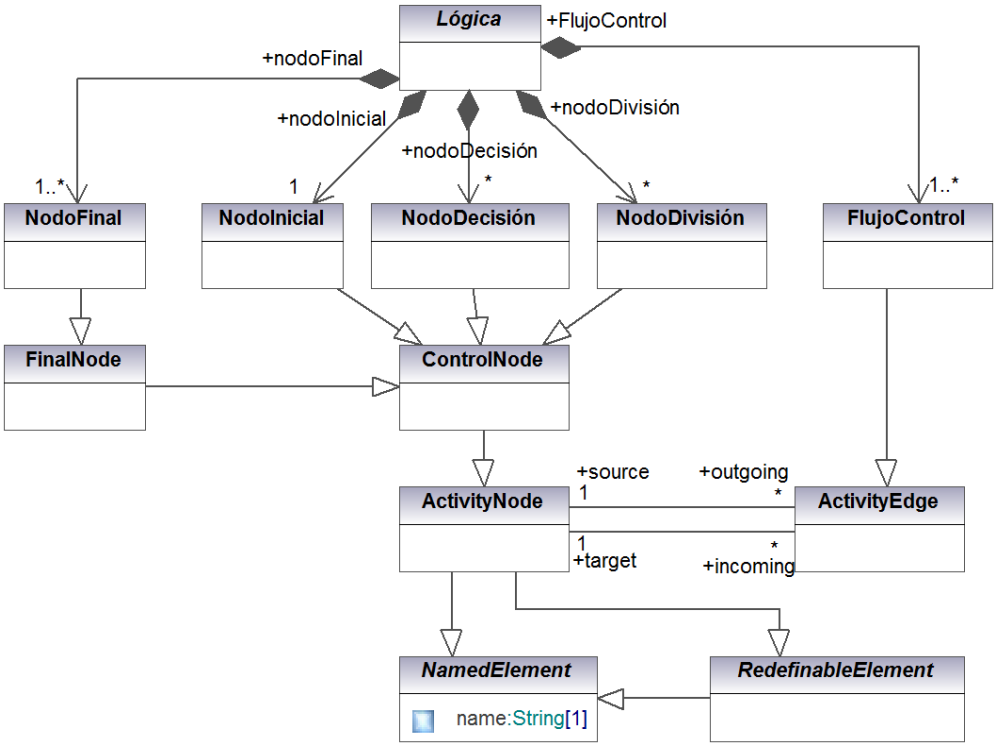


Figura 3-8: Detalle de la Vista del Experto de Dominio del meta-modelo CADAMM: diagrama de actividades (elemento *Lógica*)

El meta-modelo de esta vista de la aproximación CADAMM se completa con las reglas de composición mostradas en la Tabla 2-2.

Tabla 3-2: Reglas de composición de la Vista del Experto de Dominio (VED) del meta-modelo CADAMM

REGLA	DESCRIPCIÓN
VED_1	El nombre de un escenario es único dentro de un sistema.
VED_2	El nombre de una aplicación (aplicación atómica o súper-aplicación) es único dentro de un escenario.
VED_3	El nombre de un componente es único dentro de una aplicación atómica.
VED_4	El nombre de un dato es único dentro de un componente.
VED_5	El número de conectores de datos debe ser como mínimo igual al número de componentes menos uno.
VED_6	Los puertos de entrada y salida, propiedades desde y hacia de un conector de datos, deben pertenecer a componentes diferentes.

REGLA	DESCRIPCIÓN
VED_7	En el diagrama de actividades de la lógica de salida, únicamente puede haber un nodo inicial, que debe ser el componente al que está asociada la lógica.
VED_8	En el diagrama de actividades de la lógica de salida, los nodos finales no pueden repetirse, y deben corresponderse con componentes siguientes (aquéllos hacia los que se dirigen conectores de datos que parten desde el puerto de salida).
VED_9	En el diagrama de actividades de la lógica de salida, los flujos de control se basan en expresiones formadas por combinaciones lógicas de los datos de salida del componente.
VED_10	En el diagrama de actividades de la lógica de entrada, únicamente puede haber un nodo inicial, que debe ser el puerto de entrada al que está asociada la lógica.
VED_11	En el diagrama de actividades de la lógica de entrada, los nodos finales no pueden repetirse, y deben corresponderse con conectores de datos que se dirigen hacia el puerto de entrada.
VED_12	En el diagrama de actividades de la lógica de entrada, los flujos de control se basan en expresiones formadas por combinaciones lógicas de las entradas del puerto de entrada.
VED_13	Toda salida de un puerto de salida proviene de un dato de salida del componente.
VED_14	Todo dato de salida de un componente no tiene que corresponderse con una salida de su componente.
VED_15	Todo dato de entrada de un componente proviene de una entrada de su puerto de entrada.
VED_16	Toda entrada de un puerto de entrada no tiene que corresponderse con un dato de entrada del componente.
VED_17	El nombre de un grupo de QoS debe ser único dentro de una aplicación.
VED_18	Un mismo grupo de QoS no puede caracterizar a una aplicación y a un componente.
VED_19	Una súper-aplicación siempre debe tener un grupo de QoS para sus requisitos temporales.
VED_20	Si una súper-aplicación contiene aplicaciones atómicas activadas por tiempo, el periodo de dichas aplicaciones debe ser inferior al de la súper-aplicación.
VED_21	Un componente no puede tener el mismo tipo de QoS agrupado en diferentes grupos de QoS.

REGLA	DESCRIPCIÓN
VED_22	Si un grupo de QoS no es flexible, debe quedar caracterizado únicamente por tipos de QoS. Además, el nombre del tipo de QoS debe ser único dentro del grupo de QoS.
VED_23	Si un grupo de QoS es flexible, debe quedar caracterizado únicamente por niveles de QoS. Cada nivel de QoS contempla una serie de tipos de QoS.
VED_24	Un grupo de QoS flexible sólo puede tener, como máximo, un nivel de QoS inicial.
VED_25	El orden y nombre de los niveles de QoS deben ser únicos dentro de un grupo de QoS flexible.
VED_26	El orden de los niveles de QoS dentro de un grupo de QoS flexible deben ser secuenciales crecientes.
VED_27	Los niveles de QoS se ven reflejados únicamente en el valor de los tipos de QoS. Esto implica que todos los niveles presentan los mismos tipos (nombre y unidades) pero con diferente valor.
VED_28	El componente encargado de realizar el cambio de nivel de un grupo de QoS flexible, no puede serlo de otros grupos. Además dicho grupo debe haberse definido en su aplicación.
VED_29	Las relaciones de tipo “afectadoPor” y “realizaCambio” se refieren a grupos de QoS flexibles definidos para la aplicación de dicho componente.
VED_30	El nombre de un evento debe ser único dentro de una aplicación.
VED_31	El nombre de un evento propagado debe ser único dentro de un escenario.
VED_32	El nombre de un evento recibido debe ser único dentro de un escenario.
VED_33	En el diagrama de actividades de la lógica de eventos, únicamente puede haber un nodo inicial, que debe ser el componente al que está asociada la lógica.
VED_34	En el diagrama de actividades de la lógica de eventos, los nodos finales no pueden repetirse, y deben corresponderse con eventos de su aplicación.
VED_35	En el diagrama de actividades de la lógica de eventos, los flujos de control se basan en combinaciones lógicas de los datos de salida del componente.
VED_36	Cada evento sólo puede ser lanzado por un único componente de la aplicación, lo que implica que únicamente puede aparecer en un diagrama de lógica de eventos.
VED_37	Aquellas aplicaciones que se quieran ejecutar con una secuencia concreta deben tener un orden secuencial.

REGLA	DESCRIPCIÓN
VED_38	Si el objetivo de una acción de tipo Crear es una súper-aplicación, no se indica estado de ejecución inicial para sus componentes.
VED_39	Si el objetivo de una acción de tipo Crear es una aplicación atómica, el estado de ejecución inicial es optativo.
VED_40	El origen de una conexión de estado tiene que ser un componente de la aplicación que lanza el evento. Del mismo modo, el destino de una conexión de estado tiene que ser un componente de la aplicación objetivo de la acción. Además, un componente no puede ser destino de varias conexiones de estado.
VED_41	Si el objetivo de una acción de tipo configurar es una súper-aplicación únicamente se puede dar valor a parámetros de QoS de la súper-aplicación.
VED_42	Una aplicación atómica que pertenece a una súper-aplicación no puede ser el objetivo de acciones de tipo crear o destruir que provienen de aplicaciones que no pertenezcan a la misma súper-aplicación.
VED_43	El origen y destino de un evento propagado (TxEventoEscn y RxEventoEscn, respectivamente) tienen que pertenecer a escenarios diferentes.
VED_44	Las acciones asociadas a un evento recibido por un escenario tienen que tener como objetivo aplicaciones que pertenecen al mismo escenario.
VED_45	El objetivo de una acción de tipo crear no puede ser la misma acción que lanzó el evento asociado.

3.4 Vista del Desarrollador de Software

El desarrollador de software es el encargado de **diseñar la solución SW** correspondiente al diseño arquitectónico definido por el experto de dominio, así como de **caracterizar y escribir el código de su implementación**. A lo largo de los siguientes sub-apartados se describen los mecanismos ofrecidos por la *Vista del Desarrollador de Software* del meta-modelo CADAMM para capturar toda esta información, durante las fases de *diseño detallado* e *implementación*.

3.4.1 Diseño Detallado

En esta fase el desarrollador diseña la solución SW, teniendo en cuenta dos aspectos que para el experto de dominio no son relevantes: preparar la siguiente fase de implementación de código y la existencia de una plataforma que gestiona la correcta ejecución de las aplicaciones. A pesar de ello, sigue siendo un diseño independiente de la tecnología, puesto que no se liga a ningún lenguaje de programación ni arquitectura de desarrollo.

A continuación se detallan los nuevos conceptos de modelado que se introducen para cumplir con este objetivo, así como los que se extienden con las particularidades del desarrollador.

3.4.1.1 Aplicación Atómica

Cuando se solicita el arranque de un escenario o súper-aplicación la plataforma de gestión es la encargada de la activación de las aplicaciones atómicas. Por lo tanto, el desarrollador debe identificar cuáles deben ser *inicialmente arrancadas* por la plataforma. Esta información es innecesaria en caso de aplicaciones activadas bajo demanda, ya que es una acción de tipo crear la que provoca su activación.

3.4.1.2 Componente

De todos los conceptos de modelado, el de componente es el único que se convierte en código fuente en la fase de implementación. Es por ello que es el principal foco del diseño detallado.

Desde el punto de vista del desarrollador, una **Unidad de Servicio** consiste en la unidad mínima de código que representa la funcionalidad de un componente, independientemente de la lógica de su aplicación. Además, dicha unidad de servicio requiere un conjunto de parámetros de entrada (concepto de **Parámetro**) para ofrecer su funcionalidad y proporciona un conjunto de parámetros de salida (Parámetro) tras su ejecución, que en última instancia serán los datos de entrada y

salida del componente que la encapsula (*refDato*). La caracterización de los parámetros de la unidad de servicio incluye el *tipo de dato*. De esta forma, se pueden validar las conexiones de datos establecidas por el experto. Únicamente se aceptan aquellas conexiones entre parámetros con el mismo tipo. Con el fin de no ligarse a un lenguaje de programación concreto, para la definición de los tipos de datos se puede hacer uso de un lenguaje de descripción de interfaces, como por ejemplo IDL (*Interface Definition Language*) (OMG, 2016), algunos de cuyos tipos básicos se muestran en la lista enumerada *TiposIDL* de la Figura 3-9.

Además, la caracterización de la unidad de servicio contiene información para guiar la generación del código del componente. En este sentido, se debe indicar si la unidad de servicio requiere acciones de *inicialización* o *finalización*. Por ejemplo, el pulsioxímetro utilizado en el demostrador descrito en el Capítulo 5 (Cooking Hacks, 2016), debe ser inicializado antes de usarlo. Esto implica que la unidad de servicio que lo gestiona precisa de tareas de inicialización. Del mismo modo, una unidad de servicio que accede a una base de datos, deberá cerrar su conexión con ella antes de finalizar su ejecución. También se debe indicar si se trata de una unidad de servicio *con estado*, y si precisa de parámetros de *configuración*. En el caso de la residencia de ancianos, por ejemplo, las unidades de servicio se pueden configurar con el identificador del paciente y/o el de su habitación asignada, información necesaria para acceder a repositorios de históricos. Lo mismo ocurre con el identificador de la sala de un banco que supervisa una aplicación de video-vigilancia, o el identificador de la región asociada a una aplicación de alerta temprana.

Por otro lado, se extiende la caracterización del **Componente** con información relevante para la plataforma de gestión. Más concretamente, se indica el *nivel de fiabilidad* de cada componente en el sentido de tolerancia a fallos en el propio componente o en el nodo en que resida en tiempo de ejecución. Se trata de una característica que representa el número de réplicas del componente que deben estar disponibles en tiempo de ejecución, con el objetivo de que la plataforma de gestión asegure la máxima disponibilidad posible (*Requisito R8* en la Tabla 3-1).

Por último, el desarrollador también debe establecer aquellas restricciones a nodo debido al uso de su plataforma hardware (HW). Tal es el caso de un recurso HW únicamente accesible desde un nodo, por ejemplo un sensor de temperatura no inteligente, y para el que todas sus instancias estarán restringidas al nodo con acceso a dicho recurso. Para ello, el desarrollador debe tener conocimiento de los recursos de la infraestructura, representados por el *modelo de recursos* en la Figura 2-1.

3.4.2 Implementación

Los mecanismos ofrecidos por la Vista del Desarrollador de Software para la fase de implementación persiguen dos objetivos principales: la reutilización y la automatización de la generación de código. Para ello, se separa la implementación de la unidad de servicio de la del componente. De hecho, este último no sólo encapsula la unidad de servicio sino también la lógica de datos y la lógica de eventos. Es decir, un componente representa una unidad de servicio formando parte de una aplicación concreta. Además, su implementación se corresponde con una determinada plataforma (lenguaje de programación, sistema operativo, librerías, etc.) y es dependiente de la plataforma de gestión de ejecución concreta, ya que incluye código para la interacción con ella. Por otro lado, en esta fase también se recoge la demanda de recursos de las diferentes implementaciones de las unidades de servicio.

3.4.2.1 Caracterización del Código de la Unidad de Servicio

Por un lado, la funcionalidad relacionada con una unidad de servicio se puede implementar de diferentes maneras (**Implementación_US**). Por otro lado, un componente puede tener varias implementaciones (**Implementación_C**), y cada una encapsulando una única implementación de la unidad de servicio correspondiente.

Se propone una interfaz común, que deben seguir todas las implementaciones de unidades de servicio, y que está formada por una serie de **Métodos** que se implementan o no en base a la información recogida en el diseño detallado. Más concretamente, los métodos definidos son los siguientes:

- *inicializar* / *finalizar*: estos métodos se implementan si en el diseño detallado se ha indicado que precisa acciones de inicialización (p. ej., configuración de un sensor) o finalización (p. ej., cierre de conexión a una base de datos). En la inicialización se incluirán, también, los parámetros de configuración señalados en el diseño detallado, tales como el identificador de la región o del paciente a supervisar.

- *procesar*: todas las unidades de servicio deben implementar este método en el que se incluye el código relativo al servicio ofrecido, es decir, su funcionalidad. Sus parámetros deben coincidir con los datos establecidos por el experto y con el tipo de dato determinado en el diseño detallado.

- *leerEstado* / *escribirEstado*: si en el diseño detallado se ha indicado que se trata de una unidad de servicio con estado se deben implementar ambos métodos, que posibilitan la transferencia indirecta del estado descrita en el capítulo del estado del arte. Así, la plataforma de gestión puede leer o actualizar el estado de ejecución de instancias del componente, respectivamente.

- *cambiarNivelQoS*: las unidades de servicio de todos aquellos componentes señalados como los responsables de realizar el cambio del nivel de QoS de una aplicación implementan este método. Sus parámetros deben coincidir con los tipos de QoS definidos para el grupo de QoS cuyo nivel se quiere modificar.

No obstante, a pesar de compartir una misma interfaz, la implementación de una unidad de servicio es dependiente del *lenguaje de programación* empleado, del *sistema operativo* y del tipo de *arquitectura* para la cual se desarrolla. En la Figura 3-9 se ilustra, a modo de ejemplo, la caracterización de las implementaciones de unidades de servicio en dos lenguajes de programación diferentes: Java como ejemplo de programación orientada a objetos (*Impl_OO_Java*, *Método_OO_Java*, *OO_Java_E* y *OO_Java_S*) y C como ejemplo de programación estructurada (*Impl_Estruct_C*, *Método_Estruct_C* y *Estruct_C_Param*).

Esta interfaz común cumple otro objetivo, independiza a las unidades de servicio de la plataforma de gestión seleccionada. De esta manera, la **reutilización** se logra a dos niveles: (1) una misma unidad de servicio se puede emplear en diferentes componentes. Por ejemplo, la unidad de servicio que toma una medida de pulso se puede emplear tanto en la aplicación *ComprobarRelajación* del *Paciente2* como en la aplicación *ControlPulso* del *Paciente3*; (2) una misma unidad de servicio se puede emplear con diferentes plataformas de gestión. Al mismo tiempo, se facilita la **generación automática** de las implementaciones de componente, que sí dependen de la plataforma de gestión. Esta generación automática la lleva a cabo el entorno de soporte propuesto en el Capítulo 4.

3.4.2.2 Demanda de Recursos

Como se ha comentado anteriormente, la plataforma de gestión necesita conocer información sobre el diseño de las aplicaciones, como por ejemplo: los diferentes escenarios y los eventos que los relacionan, las aplicaciones de cada escenario y los eventos que las relacionan, los componentes de dichas aplicaciones y sus implementaciones. Además, como dicha plataforma gestiona recursos del sistema, también necesita conocer la **demanda de recursos** de las unidades de servicio. Dicha demanda se suele expresar en función del ancho de banda de *CPU*, de la carga de *memoria*, y del *ancho de banda* de red. Además, en función de los mecanismos ofrecidos por la plataforma de gestión de la ejecución puede ser necesario expresar esta demanda en tres niveles, por lo que el modelado los contempla: 1) valores mínimos para decidir la aceptación de nuevas aplicaciones; 2) valores medios para medir cargas normales; 3) valores máximos en situaciones de peor caso.

Por otro lado, la demanda de recursos también está ligada con la correcta gestión de la calidad de servicio flexible soportada por determinadas aplicaciones. Esta degradación de QoS se traduce en variaciones en su demanda de recursos, a mayor calidad mayor demanda. Por lo tanto, el desarrollador debe especificar los recursos demandados por las implementaciones de unidad de servicio de aquellos componentes afectados por la QoS flexible, en cada uno de sus niveles de flexibilidad.

De esta manera la plataforma de gestión será capaz de adaptar la demanda de recursos de las aplicaciones a la disponibilidad del sistema en un momento concreto, modificando su nivel de calidad, asegurando así el mejor nivel de calidad posible.

3.4.3 Meta-modelo y Reglas de Composición

La Figura 3-9 muestra el meta-modelo correspondiente a la Vista del Desarrollador de Software que, como se ha comentado, es una extensión de la Vista del Experto de Dominio, por lo que incluye todos sus elementos de modelado. Sin embargo, por simplicidad únicamente se muestran aquellas propiedades y elementos nuevos (resaltados en verde en la Figura 3-9), así como aquéllos de la Vista del Experto de Dominio con los que se relacionan.

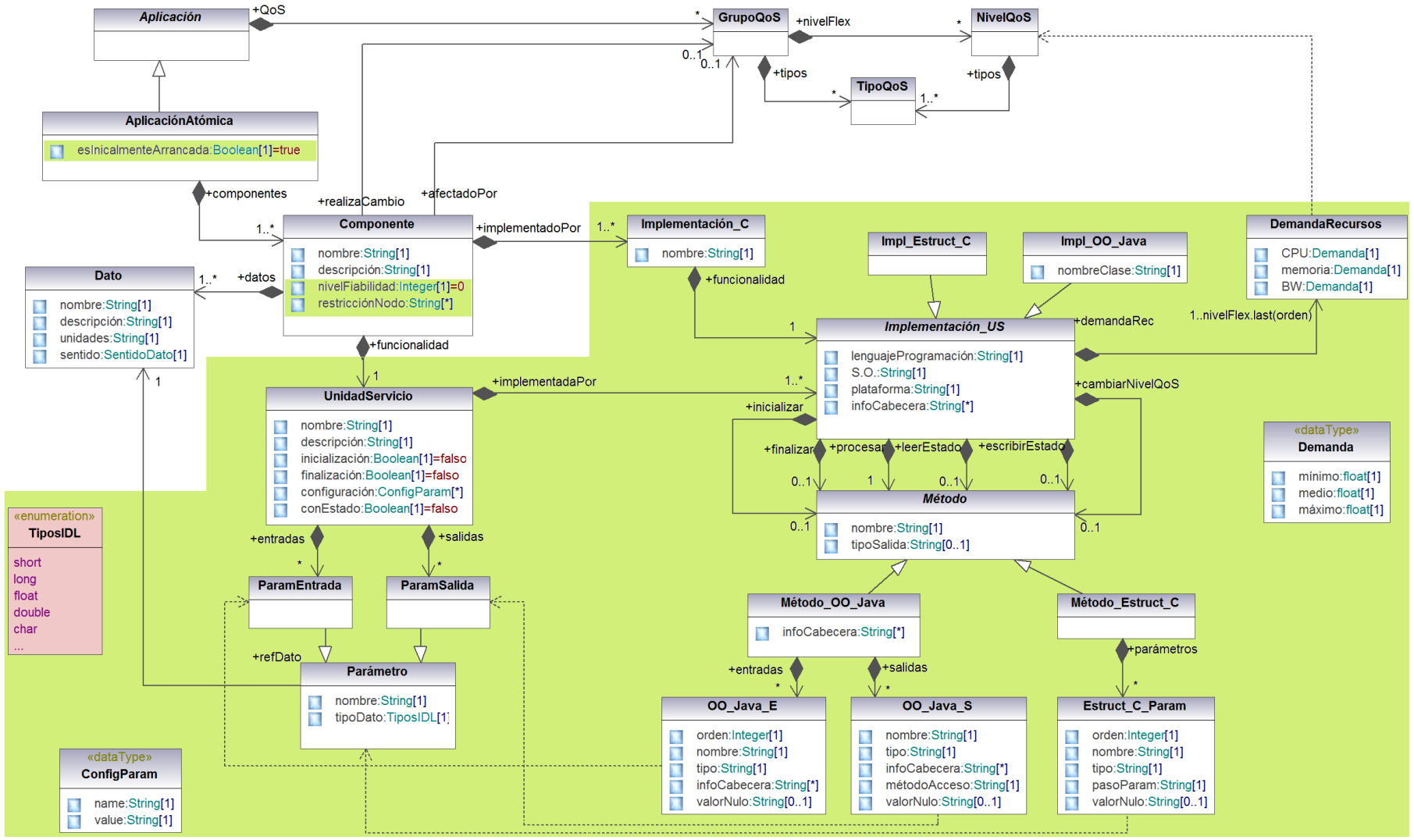


Figura 3-9: Vista del Desarrollador de Software del meta-modelo CADAMM

Esta vista se completa con las reglas de composición mostradas en la Tabla 3-3.

Tabla 3-3: Reglas de composición de la Vista del Desarrollador de Software (VDS) del meta-modelo CADAMM

REGLA	DESCRIPCIÓN
VDS_1	El tipo del parámetro de entrada y de salida que conforman una conexión de datos debe ser el mismo.
VDS_2	El número de parámetros de entrada de la unidad de servicio debe coincidir con el número de datos de entrada del componente.
VDS_3	El número de parámetros de salida de la unidad de servicio debe coincidir con el número de datos de salida del componente.
VDS_4	Si la unidad de servicio precisa tareas de inicialización, todas sus implementaciones deben contener el método inicializar.
VDS_5	Si la unidad de servicio precisa tareas de finalización, todas sus implementaciones deben contener el método finalizar.
VDS_6	Si se trata de una unidad de servicio con estado, todas sus implementaciones deben contener los métodos leerEstado y escribirEstado.
VDS_7	Si el componente es el encargado de cambiar el nivel de QoS, todas las implementaciones de su unidad de servicio deben contener el método cambiarNivelQoS.
VDS_8	Todas las implementaciones de unidad de servicio tienen una demanda de recursos asociada. Si la unidad de servicio se corresponde con un componente afectado por QoS flexible, se debe indicar una demanda de recursos por cada nivel de calidad.

3.5 Conclusiones

En este capítulo se han identificado los **requisitos comunes** de las aplicaciones distribuidas sensibles al contexto, analizando tres campos de aplicación diferentes: asistencia domiciliaria, detección temprana de catástrofes medio-ambientales y video-vigilancia.

Partiendo de esta problemática, se ha propuesto una **aproximación de modelado** para la *especificación y diseño* de este tipo de aplicaciones, proporcionando los

mecanismos necesarios para *hacer frente a los requisitos identificados*, tal y como se resume en la Tabla 3-4. Además, también proporciona mecanismos para capturar los *requisitos de ejecución* de dichas aplicaciones que afectarán a la plataforma de gestión responsable de su ejecución y que son necesarios para cumplir todos los requisitos. A este respecto resulta importante destacar que el *diseño* de las aplicaciones es *transparente al aseguramiento de su disponibilidad y seguridad*, ya que se consideran requisitos cuyo cumplimiento es total responsabilidad de la plataforma de gestión.

Tabla 3-4: Cumplimiento de los requisitos de las aplicaciones por la aproximación de modelado CADAMM y requisitos para la plataforma de gestión

REQUISITO	CONCEPTOS DE MODELADO	REQUISITOS PLATAFORMA DE GESTION
R1 – Distribución de aplicaciones (Integración de sensores y equipos de procesamiento distribuidos sobre plataformas heterogéneas)	Componente, Puerto de Entrada, Puerto de Salida, Conector de Datos.	Ejecución concurrente de instancias de componente en dispositivos de procesamiento heterogéneos. Soporte a la comunicación entre dispositivos e instancias de componente.
R2 – Adquisición, procesamiento y actuación personalizados (Adecuación de las tareas de adquisición, procesamiento y actuación a las particularidades del campo de aplicación y del entorno bajo monitorización concreto)	Escenario, Súper-Aplicación, Aplicación Atómica, Unidad de Servicio, Componente, Dato, Entrada, Salida, Conector y Conexión de Datos, Lógica de Entrada y Salida, Grupo y Tipo de QoS.	
R3 – Escalabilidad (Número y tamaño de aplicaciones variable y manejable. Soporte a la modificación (inclusión / eliminación) de aplicaciones y recursos compartidos)	Escenario, Súper-Aplicación, Aplicación Atómica.	Soporte al registro de aplicaciones / súper-aplicaciones/escenarios y sus características. Soporte al registro de recursos (altas/bajas).
R4 – Calidad de servicio específica de aplicación (Soporte a la especificación y aseguramiento de la calidad de servicio demandada por cada tipo de aplicación)	Grupo de QoS, Tipo de QoS.	
R5 – Calidad de servicio flexible (Capacidad para especificar y gestionar los distintos niveles de demanda de recursos soportados por una aplicación)	Grupo de QoS, Tipo de QoS, Nivel de QoS.	Soporte al ajuste de calidad de servicio en aplicaciones para cumplir requisitos de ejecución, de forma transparente a las aplicaciones.

REQUISITO	CONCEPTOS DE MODELADO	REQUISITOS PLATAFORMA DE GESTION
R6 – Tipos de activación (Tareas de medida o procesamiento activadas por tiempo o bajo demanda)	Lógica de Entrada, Lógica de Salida, Grupo de QoS, Tipo de QoS, Evento, Acción.	Ejecución síncrona o por recepción de datos de instancias de componente.
R7 – Adaptabilidad (Sensibilidad a cambios en el contexto: lanzamiento/parada de aplicaciones, modificación de sus propiedades…)	Evento, Puerto de Eventos, Lógica de Eventos, Acción, Grupo de QoS, Tipo de QoS, Nivel de QoS, TxEventoEscn, RxEventoEscn.	Soporte a la interacción instancia-plataforma para la gestión de las acciones a ejecutar sobre aplicaciones, en respuesta a un evento de aplicación.
R8 – Disponibilidad (Asegurar la continuidad del servicio, o la restauración tan pronto como sea posible sin afectar a las aplicaciones)	Especificación del nivel de fiabilidad.	Aseguramiento de la tolerancia a fallos de componente/nodo, teniendo en cuenta el nivel de fiabilidad, de forma transparente a las aplicaciones, incluso para aquellas con estado.
R9 – Seguridad (Asegurar la privacidad, confidencialidad e integridad de los datos gestionados por las aplicaciones)		Aseguramiento del control en el acceso a datos y de la confidencialidad de los mismos.

Se han definido dos **vistas de dominio** de acuerdo a los diferentes expertos que participan en dicha especificación y diseño. Así, los conceptos de modelado identificados en la **Vista del Experto de Dominio** permiten al *experto de dominio especificar el diseño arquitectónico de las aplicaciones abstrayéndolo de los detalles de la tecnología* (implementación y plataforma de gestión). Por otro lado, la **Vista del Desarrollador de Software** proporciona mecanismos para que el *desarrollador de software* realice el *diseño de la solución SW* correspondiente a la especificación del experto de dominio *sin ligarse a ninguna tecnología*. Esta vista también facilita que el desarrollador de software *caracterice el código* correspondiente a la implementación de las unidades de servicio de dicha solución SW. Por lo tanto, esta división de la aproximación de modelado en diferentes vistas de dominio permite la *colaboración entre diferentes expertos* con áreas de conocimiento muy diversas (expertos de dominio y desarrolladores de software).

Gracias a los conceptos de Componente y Unidad de Servicio, y a la división del proceso de generación de código en dos fases (generación de la implementación de unidad de servicio y generación de la implementación de componente), la solución propuesta favorece la *reutilización de código* de una misma unidad de servicio en diferentes aplicaciones e incluso en diferentes plataformas de gestión, ya que no es hasta la generación del código de la implementación de componente donde se liga a una plataforma de gestión de la ejecución concreta. Además, este hecho también favorece la *automatización del proceso de generación de código* que lleva a cabo el entorno de soporte propuesto en el Capítulo 4.

Por último, resulta importante destacar que la aproximación de modelado recogida en el meta-modelo CADAMM se ha definido de tal manera que: (1) puede ser personalizada a las particularidades de un campo de aplicación concreto (obteniendo así el meta-modelo *MM_Aplicaciones* de la Figura 2-1); y (2) puede ser extendida para incorporar nuevas demandas de otros ámbitos no analizados en este trabajo, como por ejemplo nuevas calidades de servicio específicas de aplicación.

4 ENTORNO DE DESARROLLO INTEGRADO: CADAMTOOLSUITE

"All sorts of things can happen when you're open to new ideas and playing around with things"

(Stephanie Kwolek)

4.1 Arquitectura de CADAMToolSuite

Este capítulo está orientado a la definición del entorno de desarrollo integrado (*Integrated Development Environment*, IDE) **CADAMToolSuite** que, basándose en la aproximación de modelado propuesta en el Capítulo 3, da **soporte al ciclo de desarrollo de aplicaciones distribuidas sensibles al contexto** (aplicaciones a partir de ahora), desde su especificación hasta su puesta en marcha, **automatizándolo**, hasta donde sea posible. El IDE debe interactuar con *diferentes usuarios* en las *distintas fases del ciclo de desarrollo*, **abstrayéndoles de la tecnología subyacente**. Además, debe proporcionar los mecanismos necesarios para que cada usuario realice sus tareas, de forma **guiada** y con un **lenguaje cercano a su área de conocimiento**, con dos objetivos principales: obtener el *modelo de aplicaciones* y generar el *código fuente* de las mismas.

Más concretamente, el IDE debe cumplir los objetivos identificados en la Tabla 4-1.

Tabla 4-1: Objetivos del IDE CADAMToolSuite

OBJETIVO	DESCRIPCIÓN
1	**Guiar en la especificación y diseño de las aplicaciones**, asegurando que se trata de un diseño *correcto e independiente* de la tecnología subyacente, lo cual favorece la *reutilización* en diferentes plataformas de gestión.
2	**Automatizar** el proceso de **generación de código** de las aplicaciones, liberando al desarrollador de la ejecución de tareas repetitivas propensas a error.
3	**Encapsular y automatizar la interacción con la plataforma de gestión**, *abstrayendo* a los usuarios de sus detalles de implementación.
4	**Facilitar** al operador la **monitorización de las aplicaciones y los recursos de sistema** en tiempo de ejecución, de forma independiente de la plataforma de gestión empleada.

Para lograr dichos objetivos, se propone un **entorno integrado** por un conjunto de módulos que no sólo interactúan con los usuarios identificados, sino que también colaboran entre ellos. El punto de partida es la aproximación de modelado CADAMM,

representada por su particularización a campo de aplicación *MM_Aplicaciones*, para la cual se propone (1) una *interfaz gráfica* que guía la especificación y diseño de las aplicaciones, y (2) el uso de *transformaciones modelo-texto* (M2T) para automatizar la generación de código.

Más concretamente, la Figura 4-1 muestra la arquitectura propuesta para cumplir con los objetivos identificados, en la que se observan los diferentes módulos que la constituyen, la colaboración entre ellos y la interacción con el exterior. También se presentan los objetivos que ayuda a cumplir cada módulo, enmarcándolos en círculos.

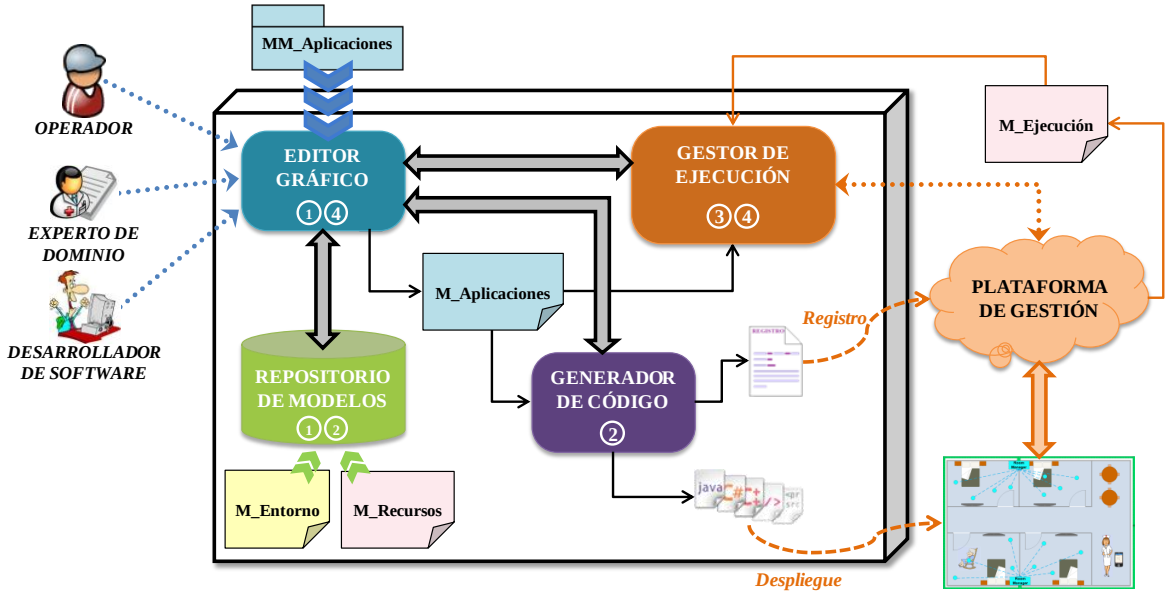


Figura 4-1: Escenario general y arquitectura del IDE CADAMToolSuite

El **Repositorio de Modelos** permite almacenar y actualizar diversos tipos de datos. Por un lado, los diferentes modelos que se manejan en el IDE: el de los recursos de la infraestructura (*M_Recursos*), el del entorno físico (*M_Entorno*), el modelo de las aplicaciones (*M_Aplicaciones*) y el modelo de ejecución (*M_Ejecución*). Por otro lado, también permite guardar la caracterización de las unidades de servicio desarrolladas. Estas unidades de servicio son identificadas por el experto de dominio, pero definidas en detalle por el desarrollador. El repositorio de modelos es un módulo independiente del campo de aplicación y plataforma de gestión.

El **Editor Gráfico** permite una *especificación y diseño guiados y correctos de las aplicaciones*. Para ello, proporciona una *interfaz gráfica amigable* que permite a cada usuario realizar sus tareas empleando una terminología cercana a su área de conocimiento, asegurando que se construye un *modelo de aplicaciones (M_Aplicaciones) correcto* conforme al meta-modelo MM_Aplicaciones. Se trata, por lo tanto, de un módulo dependiente del campo de aplicación. Por otro lado, el Editor Gráfico encapsula el acceso al resto de módulos, por lo que interactúa con todos ellos, compartiendo información a través del modelo *M_Aplicaciones*.

El objetivo principal del módulo **Generador de Código** es *automatizar* el proceso de *generación de código de las implementaciones de componentes* para una plataforma de gestión concreta, partiendo del modelo *M_Aplicaciones* especificado. Teniendo en cuenta que para la generación de código es imprescindible conocer los identificadores asignados por la plataforma de gestión a los elementos del modelo, este módulo también se encarga de realizar el registro de la información de diseño en dicha plataforma. El código generado, además de la invocación a la implementación de unidad de servicio, tiene en cuenta la interconexión de datos (lógica de entrada y lógica de salida), la interacción entre diferentes aplicaciones (lógica de lanzamiento de eventos) así como la interacción con la plataforma de gestión (tareas de control como la gestión del estado o el cambio de nivel de QoS). Todo ello haciendo uso de mecanismos ofrecidos por la plataforma de gestión en forma de API.

Por último, el **Gestor de Ejecución** controla la interacción con la plataforma de gestión durante la fase de ejecución, abstrayendo de sus particularidades a los diferentes usuarios del IDE. Dicha interacción se produce en dos situaciones diferentes:

1. Solicitud de *arranque/parada* de la ejecución de aplicaciones.
2. *Gestión del modelo de ejecución* proporcionado por la plataforma de gestión. Dicho modelo debe transformarse en información que el *Operador* pueda visualizar en el Editor Gráfico, independientemente de la plataforma concreta.

Como resultado de la interacción entre los módulos del IDE se **soporta el ciclo de desarrollo** de las aplicaciones, resumido en los siguientes pasos:

1) *Especificación y* diseño que consiste en:

    a. *Diseño arquitectónico* por parte del experto de dominio que captura los requisitos de las aplicaciones, a través del *Editor Gráfico* y apoyado en el *Repositorio de Modelos*.

    b. *Diseño detallado* por parte del desarrollador, a través del *Repositorio de Modelos* y apoyado en el *Editor Gráfico*.

    c. *Diseño de las implementaciones de las unidades de servicio* por parte del desarrollador, a través del *Repositorio de Modelos* y apoyado en el *Editor Gráfico*.

    Si las unidades de servicio necesarias ya existen en el repositorio, se reutilizan, automatizando parte de este proceso. Lo mismo ocurre con sus implementaciones.

2) *Registro* en la plataforma de gestión a través del *Generador de Código*.

3) *Generación automática de código* de las implementaciones de componentes, mediante el *Generador de Código*, haciendo uso de los identificadores proporcionados por la plataforma de gestión tras el registro.

4) *Despliegue* del código generado.

5) *Puesta en marcha* de las aplicaciones, a través del *Gestor de Ejecución*.

A lo largo de los diferentes apartados de este capítulo se describe cada uno de los módulos de CADAMToolSuite. Resulta importante destacar que en lo referente al diseño de los diferentes módulos se han perseguido dos objetivos: (1) reducir al mínimo los costes de la personalización a campo de aplicación y/o plataforma de gestión, al mismo tiempo que (2) proporcionar una interfaz sencilla y amigable para la interacción con los diferentes usuarios, abstrayéndoles de las particularidades de la plataforma de gestión. Con respecto a su desarrollo, se ha optado por la plataforma *Eclipse Modeling Project* (**EMP**) (Gronback, 2009). Se trata de una de las plataformas

más utilizadas para el desarrollo de soluciones basadas en MDE, debido al extenso conjunto de tecnologías de apoyo al desarrollo que proporciona. Además, EMP está basado en Eclipse lo que permite sacar provecho del diseño modular del propio Eclipse, que se construye componiendo componentes llamados *plug-ins*. Cada plug-in contribuye al conjunto con una determinada funcionalidad, pudiendo depender de servicios proporcionados por otros plug-ins, al mismo tiempo que otros pueden depender de los que él proporciona. Por lo tanto, se puede organizar CADAMToolSuite en forma de plug-ins de Eclipse, un plug-in por cada módulo, definiendo colaboraciones entre ellos.

4.2 Repositorio de Modelos

El Repositorio de Modelos es un módulo de apoyo al resto de módulos del entorno CADAMToolSuite, ya que permite **almacenar** los **modelos** y datos que gestionan. Más concretamente, en el Repositorio de Modelos se guarda información relativa tanto al diseño como a la ejecución de las aplicaciones del sistema, siguiendo la arquitectura propuesta en la Figura 4-2:

- *Modelo de recursos* de la infraestructura: contiene la descripción de la plataforma SW y HW de los recursos del sistema, lo que incluye la caracterización y acceso a los diferentes sensores y actuadores disponibles. Se trata de un modelo de apoyo al que el Editor Gráfico accede con el objetivo de extraer la información necesaria para establecer las restricciones a nodo de los componentes de las aplicaciones. Por otro lado, como puede haber altas y bajas de recursos se debe permitir actualizar dicho modelo.
- *Modelo del entorno físico*: representa la distribución del entorno y su instrumentación (sensores y actuadores). Se trata de un modelo de apoyo al que el Editor Gráfico accede con el objetivo de extraer la información necesaria para la especificación de escenarios y aplicaciones. Además, se debe permitir la actualización de dicho modelo en caso de que se produzca algún cambio, principalmente en la instrumentación del entorno.

- *Modelo de aplicaciones*: contiene la especificación de las aplicaciones definida mediante el Editor Gráfico que también proporciona su correspondiente representación gráfica. Se debe permitir guardarlo así como recuperar especificaciones anteriores para modificarlas.
- *Unidades de servicio e implementaciones*: aquéllas ya identificadas y/o desarrolladas que se usan para especificar la funcionalidad de las aplicaciones.
- *Modelos de ejecución*: se refieren al *estado de ejecución* del sistema en un instante concreto y al *histórico de eventos*. Su almacenamiento permite que el operario realice análisis de datos posteriores.

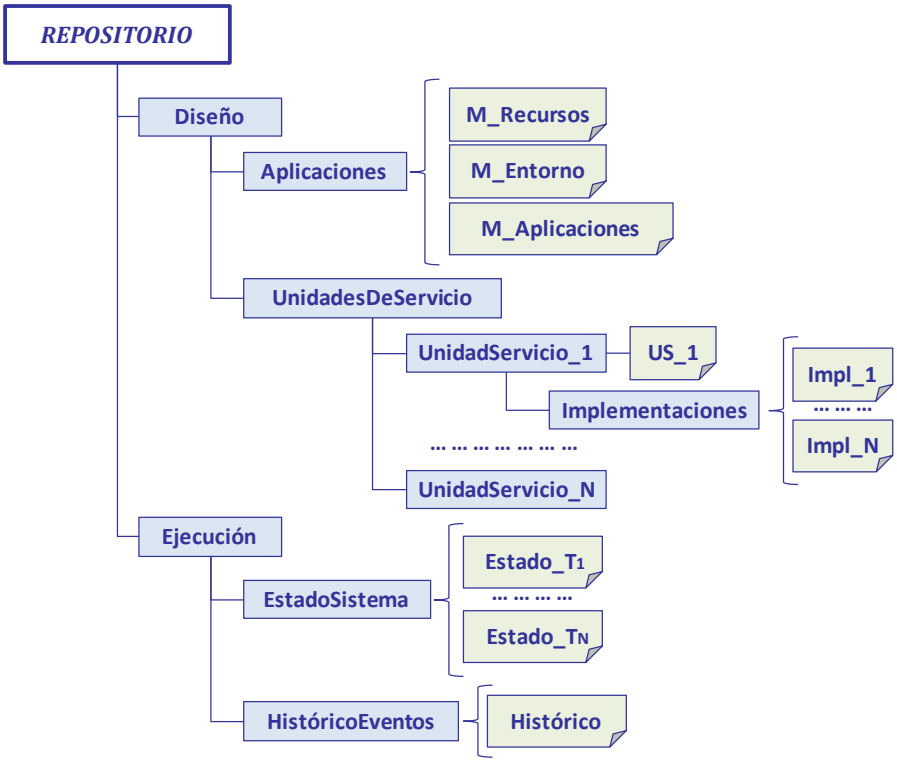


Figura 4-2: Arquitectura del Repositorio de Modelos

Para la gestión del Repositorio de Modelos se ha definido la **API** mostrada en la Tabla 4-2, que se ha desarrollado como un plug-in de Eclipse para que el Editor Gráfico pueda acceder a este módulo.

Tabla 4-2: API del Repositorio de Modelos

MÉTODO	DESCRIPCIÓN
guardarMR	Guarda el modelo de recursos que se encuentra en la ruta indicada. `void guardarMR (String rutaModelo)`
actualizarMR	Reemplaza el modelo de recursos almacenado, *M_Recursos*, con el que se encuentra en la ruta indicada. `void actualizarMR (String rutaNuevoModelo)`
extraerNodos	Extrae los nodos disponibles del modelo de recursos almacenado (*M_Recursos*). `String[] extraerNodos()`
guardarMEF	Guarda el modelo del entorno físico que se encuentra en la ruta indicada. `void guardarMEF (String rutaModelo)`
actualizarMEF	Reemplaza el modelo del entorno físico almacenado, *M_Entorno*, con el que se encuentra en la ruta indicada. `void actualizarMEF (String rutaNuevoModelo)`
extraerInfoEntorno	Extrae del modelo del entorno físico almacenado, *M_Entorno*, la información necesaria para la especificación de escenarios y aplicaciones. `Object extraerInfoEntorno()`
guardarME	A partir del modelo de ejecución recibido, actualiza el histórico de eventos (*Histórico*) y guarda el nuevo estado del sistema (*Estado_*T_N). `void guardarME (Document nuevoModelo)`
recuperarHistorico	Recupera el histórico de eventos almacenado (*Histórico*). `Document recuperarHistorico()`

recuperarEstado	Agrupa en un único modelo todos los correspondientes a los estados del sistema entre los dos instantes indicados (T_i-T_f).
	`Document recuperarHistorico (Date Ti,`
	`Date Tf)`

guardarMA	Guarda el modelo de aplicaciones y su representación gráfica.
	`void guardarMA (Document nuevoModelo,`
	`Document nuevoGrafico)`

recuperarMA	Recupera el modelo de aplicaciones almacenado (*M_Aplicaciones*). `Document[] recuperarMA()`

MÉTODO	DESCRIPCIÓN
buscarUS	Busca, en base a palabras clave, una unidad de servicio. `Document buscarUS (String[] palabrasClave)`
guardarUS	Guarda una nueva unidad de servicio. `void guardarUS (Document nuevaUS)`
actualizarUS	Actualiza la caracterización de una unidad de servicio almacenada. `void actualizarUS (Document nuevaUS)`
buscarImplUS	Busca, en base a palabras clave, una implementación de unidad de servicio. `Document buscarImplUS(String[] palabrasClave)`
guardarImplUS	Guarda una nueva implementación de una unidad de servicio almacenada. `void guardarImplUS (Document nuevaImpl)`

Teniendo en cuenta que los modelos se almacenan con formato XML, este módulo ha sido implementado mediante la base de datos nativa XML **eXist** (Siegel & Retter, 2014), que además de gestionar documentos XML permite realizar búsquedas en ellos. En eXist los datos se almacenan como una colección jerárquica de documentos, de manera que cada una de las ramas de la arquitectura propuesta se corresponde con una colección. Además permite almacenar documentos con diferentes estructuras dentro de la misma colección, lo cual proporciona la flexibilidad necesaria para almacenar juntos modelos tan dispares como M_Entorno, M_Recursos y M_Aplicaciones. Por último, eXist proporciona mecanismos para la actualización y borrado de partes de un documento XML, muy útiles para la actualización de las unidades de servicio así como del histórico de eventos.

4.3 Editor Gráfico

El Editor Gráfico es el *módulo central* de CADAMToolSuite ya que es el responsable de la *colaboración con el resto de módulos*, y debido a que los diferentes *usuarios interactúan* con el IDE a través de él. No obstante, este apartado únicamente se centra en los mecanismos ofrecidos para guiar la especificación y diseño de las aplicaciones.

Para su desarrollo se ha empleado el framework **GMF** (Eclipse, 2010a), un sub-proyecto de EMP para la definición de *lenguajes de modelado de tipo gráfico*. Los lenguajes de modelado permiten una completa descripción de los sistemas o aplicaciones antes de su construcción. Se basan en meta-modelos que constituyen la *sintaxis abstracta* del lenguaje de modelado, que se completa con su *sintaxis concreta*, la cual define la notación empleada (textual o gráfica). De hecho, GMF proporciona una plataforma para construir editores gráficos a partir de sintaxis abstractas definidas mediante el lenguaje de meta-modelado Ecore, parte del también sub-proyecto de Eclipse llamado *Eclipse Modeling Framework* (EMF) (Steinberg et al., 2008).

El proceso de desarrollo del Editor Gráfico se muestra en la Figura 4-3 y consiste en la obtención del plug-in correspondiente a un editor básico, haciendo uso de GMF, y su posterior personalización para lograr las funcionalidades deseadas. A este respecto resulta importante señalar que la modificación de un editor basado en GMF no es una tarea sencilla y requiere de un elevado nivel de experiencia. No obstante, con la intención de aprovechar la experiencia adquirida por la autora para la programación de este tipo de editores en el marco del proyecto iLAND (García-Valls, Rodríguez-López, et al., 2013), se ha optado por hacer uso de GMF.

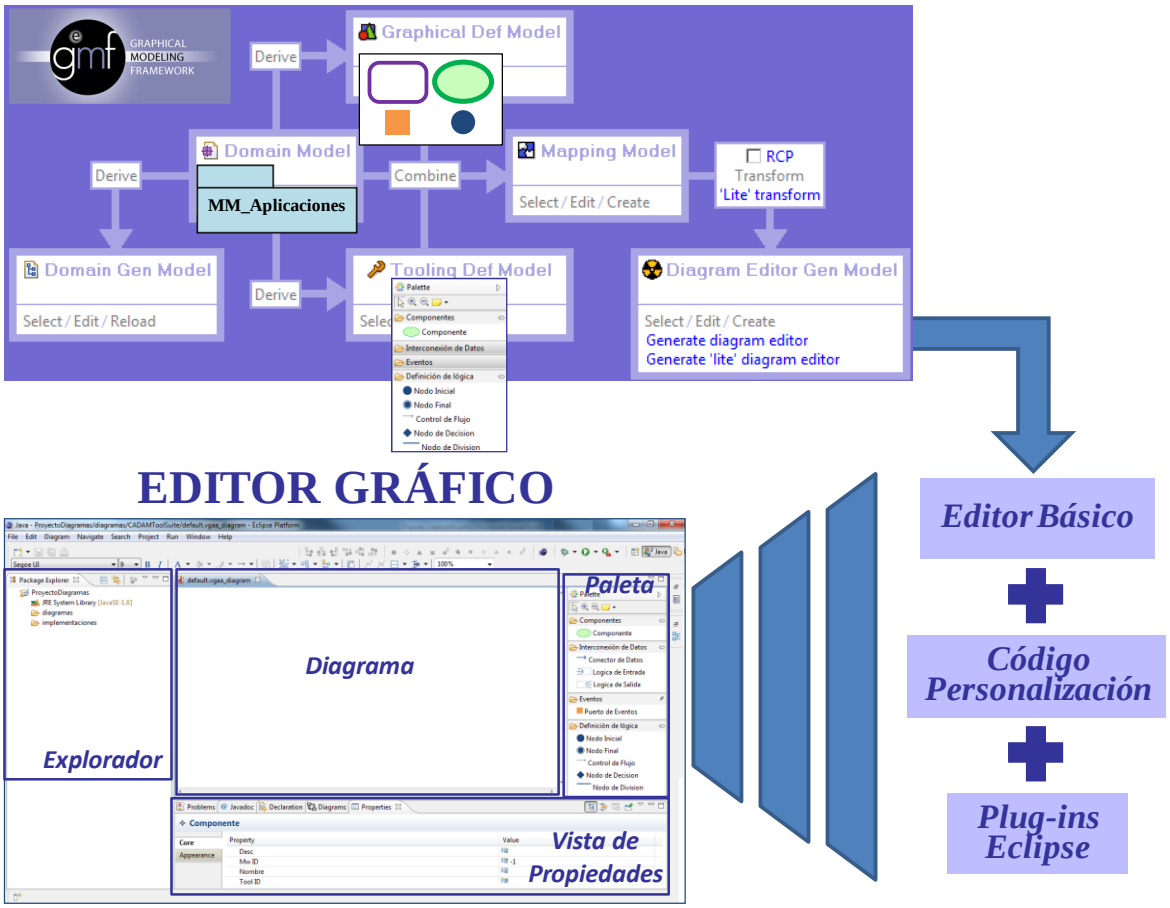


Figura 4-3: Proceso de desarrollo de editores gráficos con GMF

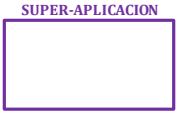
4.3.1 Editor Básico

GMF establece el proceso de desarrollo de editores gráficos, mostrado en la parte superior de la Figura 4-3, fundamentalmente basado en la definición de cuatro modelos: (1) el del dominio; (2) el de la definición gráfica; (3) el de la paleta de herramientas de creación; y (4) el de mapeo.

El núcleo de este proceso es el meta-modelo *MM_Aplicaciones* en el que se basa el IDE, que en terminología GMF se conoce como **modelo de dominio** (*Domain Model*). El **modelo de la definición gráfica** (*Graphical Definition Model*) contiene la representación gráfica para los elementos del modelo de dominio, en forma de nodos y conexiones cuya apariencia se puede determinar (forma, color, tamaño, borde, etc.). Para la especificación de este modelo se han propuesto los iconos gráficos que se muestran en la Tabla 4-3, donde también se identifican su *semántica*, que en algunos

elementos puede variar de un campo de aplicación a otro (aquéllos resaltados en color amarillo en la Tabla 4-3).

Tabla 4-3: Representación gráfica de los elementos del modelo de dominio

LÉXICO	SEMÁNTICA	ICONO
Escenario	Conjunto de actividades de supervisión que tienen algo en común. Por ejemplo, paciente en asistencia domiciliaria o región de la montaña en alerta temprana de desastres naturales.	
Súper-Aplicación	Actividad de supervisión que engloba otras, condicionando su activación.	
Aplicación Atómica	Actividad de supervisión personalizada.	
Componente	Una de las tareas de adquisición, procesamiento o actuación en las que descompone una aplicación atómica.	
Puerto de Entrada	Punto de acceso al componente por el que recibe los datos necesarios para que su unidad de servicio lleve a cabo su funcionalidad.	
Puerto de Salida	Punto de acceso al componente por el que envía los datos obtenidos como resultado de la funcionalidad de su unidad de servicio.	
Conector de Datos	Enlace entre componentes para encapsular el flujo de datos intercambiado.	
Evento	Ocurrencia de un suceso relevante que requiere una reacción.	
Puerto de Eventos	Punto de acceso al componente por el que solicita el lanzamiento de eventos.	
Acción	Reacción ante un suceso relevante para crear, destruir o modificar actividades de supervisión.	

LÉXICO	SEMÁNTICA	ICONO
Evento Propagado	Suceso relevante cuya reacción afecta a aplicaciones de otros escenarios.	
Evento Recibido	Suceso relevante ocurrido en otro escenario cuya reacción afecta a aplicaciones del escenario actual.	
Propagación de Eventos	Propagación de un suceso relevante de un escenario a otro.	
Lógica	Diagrama de actividades que define la lógica de entrada, la de salida o la de lanzamiento de eventos.	
Nodo Inicial	Inicio del diagrama de actividades que puede representar al componente actual o a su puerto de entrada.	
Nodo Final	Final de una rama del diagrama de actividades que puede representar a un conector de datos, al siguiente componente o a un evento lanzado.	
Control de Flujo	Unión entre los diferentes elementos del diagrama de actividades. Puede llevar asociada una condición que determina si el camino indicado por el elemento se sigue o no, en función de si dicha condición se cumple o no.	
Nodo de Decisión	Punto del diagrama en el que se establece una condición.	
Nodo de División	Punto del diagrama en el que se bifurcan los caminos.	

El **modelo de la definición de herramientas** (*Tooling Definition Model*) recoge el diseño de la paleta de herramientas para la creación de los elementos del modelo. Cabe destacar que no todos los elementos del modelo de dominio tienen representación gráfica, ni todos tienen herramienta de creación asociada. De hecho, es en el **modelo de mapeo** (*Mapping Model*) donde se establece la relación entre los elementos del modelo de dominio, su representación gráfica y/o su herramienta de creación. En este modelo también se pueden introducir restricciones, relacionadas

con las reglas de composición del meta-modelo MM_Aplicaciones, que permiten o evitan determinadas acciones durante la especificación de las aplicaciones.

A partir de estos cuatro modelos el framework GMF genera el código correspondiente a un **Editor Básico**, en forma de plug-in de Eclipse. Tal y como se observa en la parte inferior izquierda de la Figura 4-3, en el editor se pueden distinguir cuatro áreas principales:

1) El *diagrama* donde se muestra la representación gráfica del modelo M_Aplicaciones.
2) La *paleta de herramientas* de creación.
3) La *vista de propiedades* para la visualización y edición de los atributos de los elementos del modelo
4) El *explorador de proyectos*, propio de Eclipse, en el que se visualizan los ficheros manejados por la herramienta, agrupados en dos directorios: directorio *diagramas* para guardar los modelos creados así como sus correspondientes representaciones gráficas; y directorio *implementaciones* donde se almacena el código generado para las implementaciones de los componentes.

Por último, haciendo uso del concepto de partición de diagramas de GMF se han definido tres **vistas de edición gráfica**: Vista Gráfica del Sistema, Vista Gráfica de Escenario y Vista Gráfica de Aplicaciones Atómicas. Estas vistas posibilitan una especificación modular de las aplicaciones ya que, tal y como se muestra en la Tabla 4-4, en cada una se permite la creación, visualización y edición de determinados elementos del modelo. Nótese que se trata de vistas de edición gráfica, no de vistas de dominio, por lo que tanto expertos de dominio como desarrolladores de software tienen acceso a ellas.

Tabla 4-4: Vistas de edición gráfica

VISTA	DESCRIPCIÓN
Vista Gráfica del Sistema	Vista única que permite la identificación de los diferentes escenarios y sus relaciones, a través de la propagación de eventos. Se trabajan los conceptos de sistema, escenario, evento propagado, evento recibido por un escenario y sus acciones asociadas.
Vista Gráfica de Escenario	Una por cada escenario, para la definición de las diferentes súper-aplicaciones y aplicaciones atómicas que lo conforman. Se hace uso de los conceptos de aplicación (súper-aplicación y aplicación atómica), evento, acción (entre aplicaciones del escenario), QoS de aplicación y QoS flexible (grupo, tipo y nivel de QoS).
Vista Gráfica de Aplicación Atómica	Una por cada aplicación atómica, para su especificación. Se trabajan los conceptos de componente, dato, puerto de entrada, entrada, puerto de salida, salida, conector de datos, conexión de datos, lógica de entrada, lógica de salida, puerto de eventos, lógica de lanzamiento de eventos y QoS de componente (grupo, tipo y nivel de QoS).

4.3.2 Personalización del Editor Básico

El **Editor Gráfico de CADAMToolSuite** es una personalización del *Editor Básico* que consisten en código Java que puede ser modificado para manteniendo la misma apariencia de editor ya descrita, añadir nuevas funcionalidades y los plug-ins relativos al resto de módulos.

Más concretamente, el *Editor Básico* ha sido personalizado para añadir **servicios de valor añadido** que aseguran la construcción de **modelos correctos**, **automatizando** la especificación y diseño de aplicaciones. Con este objetivo se han definido mecanismos que *prohíben acciones* que de antemano se saben *erróneas*, o mecanismos para *proponer únicamente opciones correctas*. En general, se implementan de forma visual algunas de las restricciones y reglas de composición que establece el meta-modelo M_Aplicaciones. A continuación se describen algunos ejemplos de mecanismos de personalización:

- Menús pop-up que dan acceso a acciones sobre elementos gráficos de un diagrama. Por ejemplo para la selección del evento de aplicación del que proviene un evento propagado o para la asignación de una unidad de servicio a un componente. En concreto, la Figura 4-4 muestra el menú pop-up accesible desde un elemento componente al que se le ofrecen acciones relacionadas con su funcionalidad, implementación, registro y generación de código.

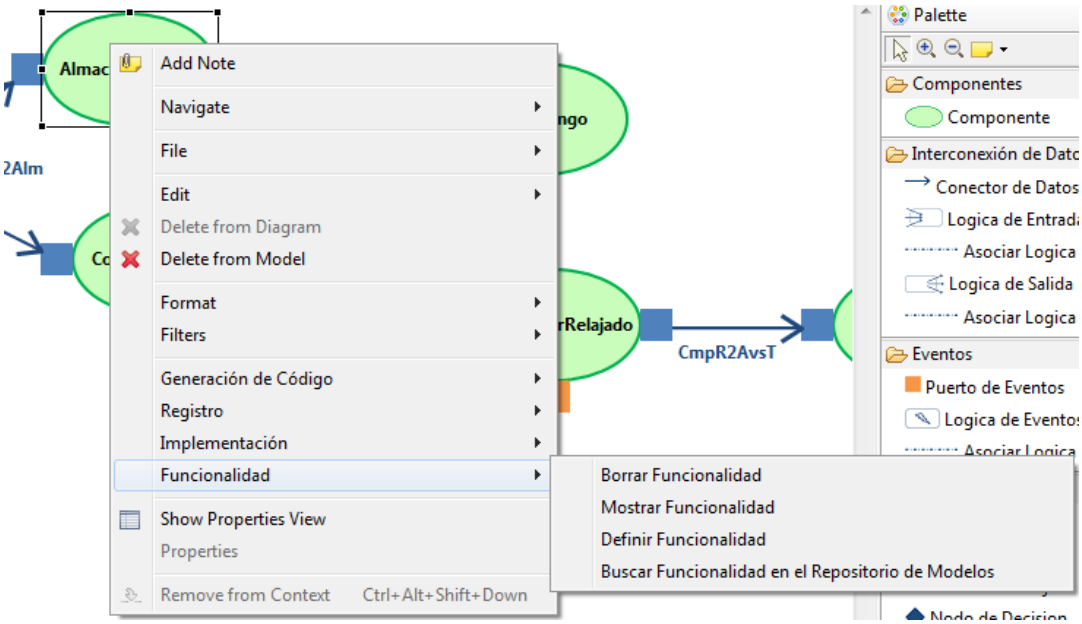


Figura 4-4: Menú pop-up: acciones sobre un componente

- Formularios que permiten la definición o selección de determinados elementos. Tal es el caso del formulario para la definición de un grupo de QoS o el formulario presentado en la Figura 4-5 para la búsqueda de unidades de servicio en base a palabras clave.

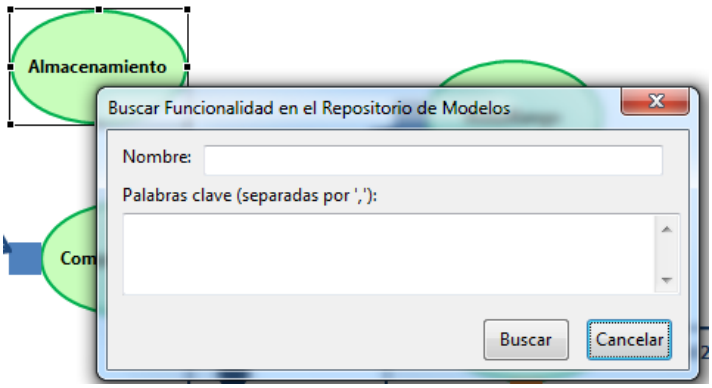
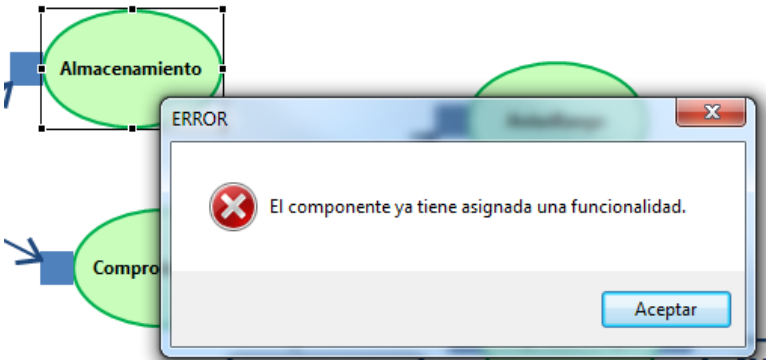
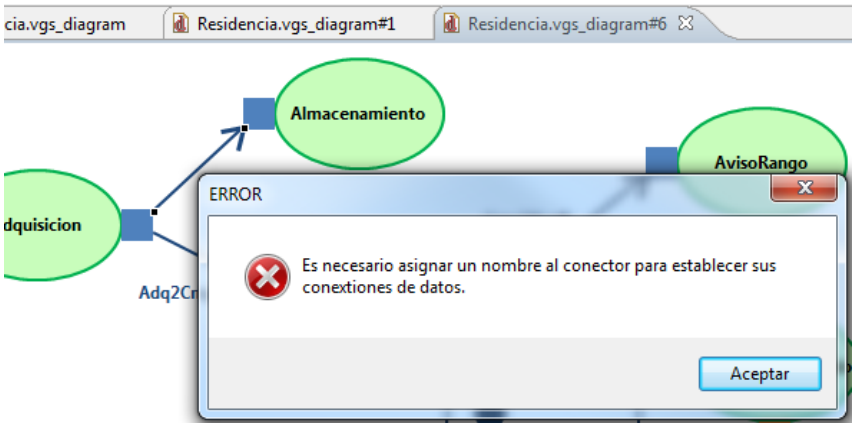


Figura 4-5: Búsqueda de unidad de servicio

- Creación automática de elementos del modelo para evitar errores de diseño, motivo por el cual no todos los elementos del modelo de dominio disponen de herramientas de creación. Así, el experto de dominio debe crear los escenarios de forma explícita haciendo uso de la correspondiente herramienta, mientras que los puertos de entrada y/o salida de un componente se crean de forma automática tras la asignación de una unidad de servicio, en función de sus entradas y/o salidas.
- Validaciones del modelo de aplicaciones o de determinadas partes: validación del sistema, validación de un escenario, validación de una aplicación. Estas validaciones permiten autorizar la ejecución de determinadas acciones. Tal es el caso del registro de una aplicación que sólo se permite si está validada.
- Gestión de las vistas de dominio. En función del identificador de usuario (experto de dominio o desarrollador), se restringe el uso de determinadas herramientas de creación así como el acceso a ciertas acciones, formularios o propiedades de elementos del modelo.
- Mensajes de aviso o error que guían a los usuarios en sus tareas. La Figura 4-6 muestra dos mensajes de error. El de la parte superior se muestra al intentar asignar una unidad de servicio a un componente al que ya se le asignó. El de la parte inferior recuerda la imposibilidad de establecer conexiones de datos en un conector de datos sin nombre.



(a)



(b)

Figura 4-6: Mensajes de error: (a) componente con funcionalidad asignada; (b) conector de datos sin nombre

4.4 Generador de Código

Este módulo se encarga de **automatizar** el proceso de **generación del código** de las aplicaciones, que al ser aplicaciones formadas por componentes que cooperan, consiste en la *generación del código* correspondiente a las *implementaciones de sus componentes*. Al tratarse de componentes que intercambian datos y de aplicaciones relacionadas por eventos lanzados como resultado de su funcionalidad, el código generado para cada implementación no sólo debe incluir la *invocación* a la *implementación de su unidad de servicio* sino también parte de la *lógica de la aplicación* y la *interacción con la plataforma* que gestiona la ejecución de las aplicaciones.

Como se ha comentado anteriormente, para la generación de código de una aplicación es imprescindible que haya sido previamente registrada en la plataforma de gestión, ya que es necesario conocer los identificadores (de aplicaciones, de componentes, de eventos…) asignados. Por último y con el objetivo de mostrar la generalidad y validez de los mecanismos ofrecidos por este módulo, en este apartado también se presenta su particularización para diferentes plataformas de gestión: una basada en componentes y otra basada en sistemas multi-agente. Se trata de dos plataformas de gestión desarrolladas en el marco de sendas tesis en curso dentro del grupo de investigación GCIS del Departamento de Ingeniería de Sistemas y Automática de la Universidad del País Vasco.

4.4.1 Registro en la Plataforma de Gestión

El registro es el proceso durante el cual se le *transfiere* a la plataforma de gestión *información acerca del diseño* de las aplicaciones. Este módulo se encarga de generar los comandos de registro acorde a dicha plataforma, extrayendo la información necesaria del modelo M_Aplicaciones, ejecutarlos y actualizar el modelo con el identificador proporcionado por la plataforma para cada elemento registrado.

Con el objetivo de asegurar la coherencia entre los elementos que se están registrando, el orden de registro debe ser acorde al de especificación (empezando por el sistema y sus escenarios, y acabando por las implementaciones de componentes), siempre teniendo en cuenta que es la estructura del repositorio de la plataforma de gestión la que determina qué elementos se deben registrar. Además, también es importante destacar que no sólo los elementos de mayor nivel deben estar registrados, sino que también el resto de elementos del mismo nivel con los que haya relación. Así, si se registra una nueva aplicación con un evento cuya acción asociada crea otra aplicación, es necesario que esta última también esté registrada. No obstante, no siempre es necesario registrar todos los elementos del modelo, tal y como ocurre al añadir una aplicación (sí se registra) a un escenario previamente registrado (no se vuelve a registrar). Por último, únicamente se permite el registro de aquellos elementos que hayan sido *previamente validados* por el Editor Gráfico.

Siguiendo con el ejemplo anterior, para poder registrar la nueva aplicación, tanto la aplicación como el escenario al que pertenece deben ser validados.

La Tabla 4-5 muestra la **API** que el Generador de Código ofrece al Editor Gráfico para llevar a cabo el registro. Cabe destacar que cada uno de estos métodos implica recorrer el modelo M_Aplicaciones desde el elemento seleccionado hasta las implementaciones de componente, siguiendo el orden de registro correcto. Nótese que también se da la posibilidad de eliminar elementos del repositorio de la plataforma de gestión.

Tabla 4-5: API del Generador de Código para el registro

MÉTODO	DESCRIPCIÓN
regSist	Accesible desde el diagrama de la Vista Gráfica de Sistema, para el registro del sistema y todos sus escenarios. `void regSist()`
regEscen	Accesible desde el diagrama de la Vista Gráfica de Escenario, para registrar el escenario y todas sus aplicaciones (súper-aplicaciones o aplicaciones atómicas). `void regEscen()`
regSuperAplic	Accesible desde una súper-aplicación de la Vista Gráfica de Escenario, para registrar la súper-aplicación y todas sus aplicaciones atómicas. `void regSuperAplic()`
regAtomica	Accesible desde el diagrama de la Vista Gráfica de Aplicación Atómica, para su registro y el de sus componentes. `void regAtomica()`
regImpl	Accesible desde un componente de la Vista Gráfica de Aplicación Atómica para registrar las implementaciones seleccionadas. `void regImpl()`
borrarEscen	Accesible desde el diagrama de la Vista Gráfica de Escenario, para eliminar el registro del escenario y todas sus aplicaciones (súper-aplicaciones o aplicaciones atómicas). `void borrarEscen()`

MÉTODO	DESCRIPCIÓN
borrarSuperAplic	Accesible desde una súper-aplicación de la Vista Gráfica de Escenario, para eliminar el registro de la súper-aplicación y todas sus aplicaciones atómicas. `void borrarSuperAplic()`
borrarAtomica	Accesible desde el diagrama de la Vista Gráfica de Aplicación Atómica, para eliminar su registro y el de sus componentes. `void borrarAtomica()`
borrarImpl	Accesible desde un componente de la Vista Gráfica de Aplicación Atómica para eliminar el registro de las implementaciones seleccionadas. `void borrarImpl()`

4.4.2 Generación Automática de Código

Se propone un proceso de generación de código que **separa la implementación del componente de la implementación de la unidad de servicio**. De hecho, se parte de que el desarrollador de software ya ha caracterizado las unidades de servicio, y ha programado y caracterizado sus implementaciones.

La Figura 4-7 muestra este proceso de generación automática de código, que está basado en el uso de transformaciones M2T, cuyo objetivo es crear texto (código) a partir de información almacenada en un modelo. Por lo tanto, el punto de partida es el **modelo de aplicaciones** (*M_Aplicaciones*) que contiene toda la información necesaria para la generación, cuyo resultado es el código (texto) correspondiente a cada una de las **implementaciones del componente**. Este código generado se debe *compilar* junto con sus correspondientes **implementaciones de unidades de servicio** (almacenadas en el repositorio de modelos) para que se pueda *desplegar* en los diferentes nodos de la infraestructura del sistema.

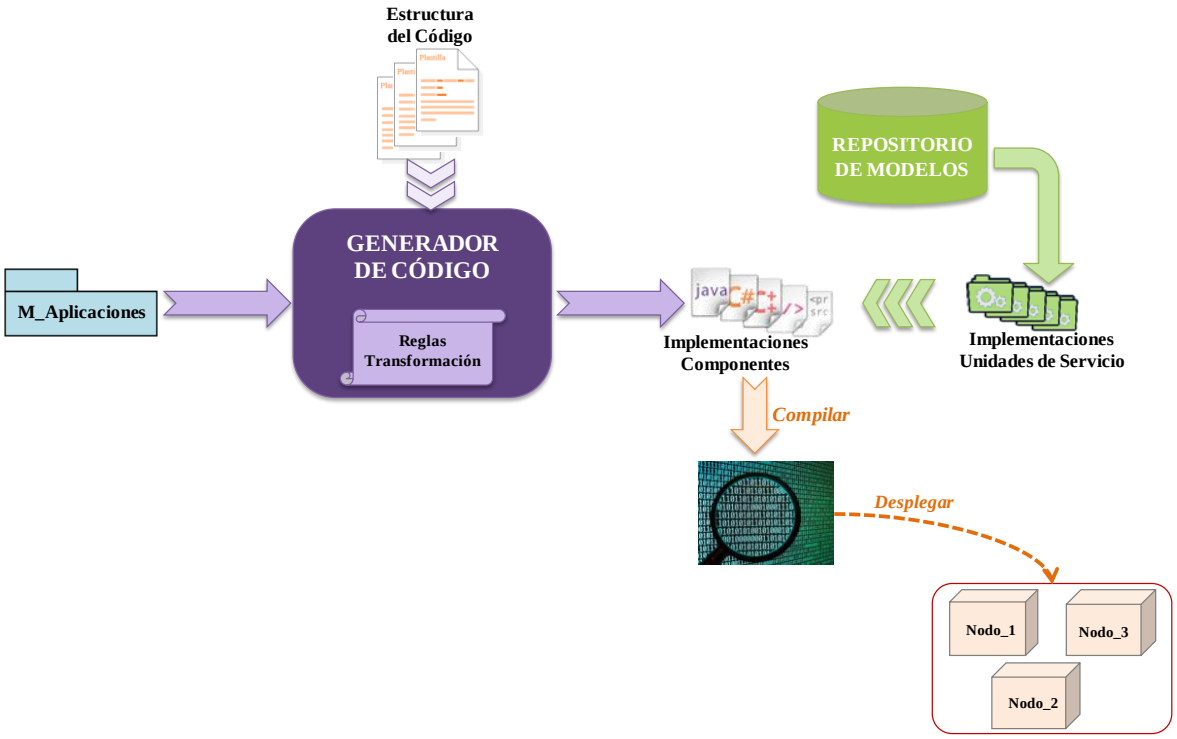


Figura 4-7: Proceso de generación automática de código de aplicaciones

Con objeto de facilitar y homogeneizar la tarea de implementación, la plataforma de gestión debe proporcionar la **estructura del código** para las implementaciones que gestiona, donde se incluyen llamadas a su API para la interacción entre las implementaciones de los componentes en ejecución (instancia a partir de ahora) y dicha plataforma. Además, esta estructura también recoge los tres estados en los que una instancia puede encontrarse: (1) inicialización, (2) ejecución y (3) finalización. El estado de **inicialización** está relacionado con la puesta en marcha de la instancia, en la que se llevan a cabo su configuración y tareas de inicialización. En el de **ejecución** la instancia ofrece su funcionalidad, de manera que en cada ciclo una instancia genérica:

1. Recibe los parámetros de entrada.
2. Invoca la ejecución de la implementación de su unidad de servicio.
3. Envía sus resultados a los componentes siguientes.
4. Lanza eventos.
5. Actualiza su estado de ejecución.

De forma paralela, en cualquier momento del estado de ejecución, la plataforma de gestión puede solicitar un cambio de nivel de QoS que la instancia debe llevar a cabo. Por último, en el estado de **finalización** se termina la ejecución de la instancia, llevándose a cabo las acciones de finalización necesarias y liberándose sus recursos.

Teniendo en cuenta la estructura del modelo de aplicaciones, el meta-modelo *MM_Aplicaciones*, así como la estructura del código a generar, se han identificado un conjunto de **reglas de transformación** que determinan cómo buscar y extraer información del modelo *M_Aplicaciones* para obtener el código correspondiente a cada una de las **implementaciones del componente**. La Tabla 4-6) muestra estas reglas de transformación, que definen a nivel componente y se aplican a cada una de sus implementaciones, obteniéndose así el código de cada estado.

Tabla 4-6: Reglas de transformación para la generación del código de implementaciones de componente

ESTADO	REGLA	DESCRIPCIÓN
Inicialización	Inicialización	Incluye las acciones de inicialización identificadas y/o identifica los parámetros de configuración y su valor.
Ejecución	Conector de Entrada	Identifica y extrae las entradas del componente recibidas a través de un conector de datos concreto, en base a sus conexiones de datos definidas.
	Puerto de Entrada	Hace uso de la regla *Conector de Entrada* para obtener los parámetros de entrada necesarios para la unidad de servicio, en función de la lógica asociada al puerto de entrada. Por lo tanto, interpreta el diagrama de actividades, si lo hay.
	Unidad de Servicio	Invoca la implementación de unidad de servicio, pasándole sus parámetros de entrada, y recopilando sus parámetros de salida.
	Conector de Salida	Identifica las salidas del componente que se envían a través de un conector de datos concreto, en base a sus conexiones de datos definidas.

ESTADO	REGLA	DESCRIPCIÓN
	Puerto de Salida	Hace uso de la regla *Conector de Salida* para definir el flujo de datos enviados a sus siguientes componentes, en función de la lógica asociada al puerto de salida. Por lo tanto, interpreta el diagrama de actividades, si lo hay.
	Acción	Identifica el tipo de acción y extrae toda la información necesaria para su ejecución: conexiones de estados o nivel de QoS, en caso de acción de tipo Crear y Configurar, respectivamente.
	Puerto de Eventos	Interpreta el diagrama de actividades correspondiente a la lógica de lanzamiento de eventos, identificando los eventos y las acciones asociadas, mediante la regla *Acción*.
	Estado de Ejecución	Actualización del estado de ejecución del componente.
	QoS Flexible	Introduce el código para realizar el cambio de nivel de QoS, mapeando los niveles de QoS de la plataforma de gestión a los niveles y tipos de QoS de la aplicación.
Finalización	Finalización	Incluye las acciones de finalización identificadas.

Para el desarrollo del Generador de Código se ha empleado **Acceleo** (Obeo Network, 2006), un *transformador M2T basado en plantillas* que también pertenece a EMP. En Acceleo, las transformaciones se basan en un conjunto de plantillas que, organizadas de forma jerárquica dentro de módulos y siguiendo un determinado meta-modelo (en este caso, MM_Aplicaciones), definen la estructura del código o texto a generar. Por lo tanto, en las plantillas se distingue una parte estática y una parte que tiene que ser sustituida por información extraída de un modelo (el modelo M_Aplicaciones) por medio de *consultas*. Además, Acceleo permite desarrollar el generador de código en forma de plug-in que se incorpora a la herramienta CADAMToolSuite y al que el desarrollador de software tiene acceso a través del Editor Gráfico. Para ello, se ha definido la API mostrado en la Tabla 4-7.

Tabla 4-7: API para la generación automática de código

MÉTODO	DESCRIPCIÓN
genEscen	Accesible desde el diagrama de la Vista Gráfica de Escenario, para generar todas las implementaciones de componentes de todas sus aplicaciones (súper-aplicaciones o aplicaciones atómicas). `void genEscen()`
genSuperAplic	Accesible desde una súper-aplicación de la Vista Gráfica de Escenario, para generar todas las implementaciones de componentes de todas sus aplicaciones atómicas. `void genSuperAplic()`
genAtomica	Accesible desde el diagrama de la Vista Gráfica de Aplicación Atómica, para generar todas las implementaciones de sus componentes. `void genAtomica()`
genComp	Accesible desde un componente de la Vista Gráfica de Aplicación Atómica, para generar determinadas implementaciones. `void genComp()`

En todos estos métodos se comprueba que todas las implementaciones a generar hayan sido previamente registradas. Además, el código obtenido se guarda en el directorio *implementaciones*, agrupado por escenarios, aplicaciones y componentes.

Teniendo en cuenta que cada una de las reglas de transformación identificadas en la Tabla 4-6 se debe implementar al menos una plantilla Acceleo, y que dicha implementación depende de la estructura de código de una plataforma de gestión concreta, los siguientes sub-apartados describen la particularización del módulo Generador de Código para dos plataformas diferentes. Ambas proporcionan los mecanismos necesarios para cumplir con los requisitos demandados por las aplicaciones de interés.

4.4.2.1 Generación de Código para la Plataforma de Gestión DAMP

DAMP (**D**istributed **A**pplications **M**anagement **P**latform) es una plataforma de gestión basada en el modelo de componentes SCA (OASIS, 2007) y que hace uso de *Data Distribution Service* (DDS) como middleware de distribución (OMG, 2007). Más

concretamente, la plataforma ha sido construida sobre la implementación Apache Tuscany de SCA (The Apache Software Foundation, 2012; Laws et al., 2011), añadiendo nuevas funcionalidades que le permiten hacer frente a las demandas de las aplicaciones de interés (Agirre et al., 2016; Agirre et al., 2015).

La plataforma DAMP gestiona un repositorio donde almacena información acerca del diseño de las aplicaciones con la estructura que se muestra en la Figura 4-8. En la parte derecha de esta figura también se muestra la relación entre los elementos del repositorio y los de la aproximación CADAMM, y que son, justamente, los elementos del modelo M_Aplicaciones que se deben registrar. Para ello, DAMP ofrece el siguiente API formado por tres métodos:

1) Registro de aplicación atómica:

   ```
   String installApp (String name)
   ```

   donde *name* es el nombre de la aplicación y devuelve el identificador asignado.

2) Registro de componente:

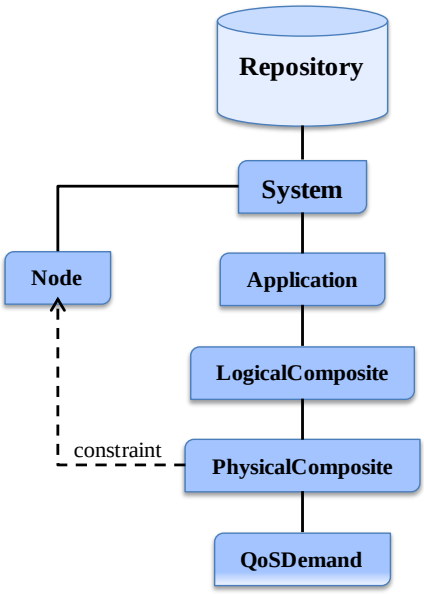
   ```
   String installLogicalComposite (String appID, String
   name, int T, int D, int P)
   ```

   donde *appID* es el identificador de su aplicación atómica, *name* es el nombre del componente y *T*, *D* y *P* son sus tipos de QoS.

3) Registro de implementación de componente:

   ```
   String installPhysicalComposite (String LcID, String
   name, String nodeID, int QoS)
   ```

   donde *LcID* es el identificador de su componente, *name* es el nombre de la implementación, *nodeID* el identificador del nodo en el que se va a desplegar y *QoS* es su demanda de recursos, que puede ser variable.



ELEMENTO REPOSITORIO	ELEMENTO CADAMM
System	Sistema
Application	AplicaciónAtómica
LogicalComposite	Componente
PysicalComposite	Implementación_C
QoSDemand	DemandaRecursos

Figura 4-8: Estructura del repositorio de la plataforma DAMP y relación con CADAMM

En cuanto a la generación de código, DAMP proporciona dos plantillas de componentes SCA en función del tipo de activación del componente: (1) *ComponentControl_T* para componentes periódicos y (2) *ComponentControl_A* para componentes activados bajo demanda. Cada plantilla constituye la clase base para la generación de código que implementa la interfaz *IControl*. Dicha interfaz proporciona todos los métodos necesarios para que la plataforma DAMP se comunique con las instancias. A modo de ejemplo, la Figura 4-9 muestra la clase base correspondiente a la plantilla *ComponentControl_A*.

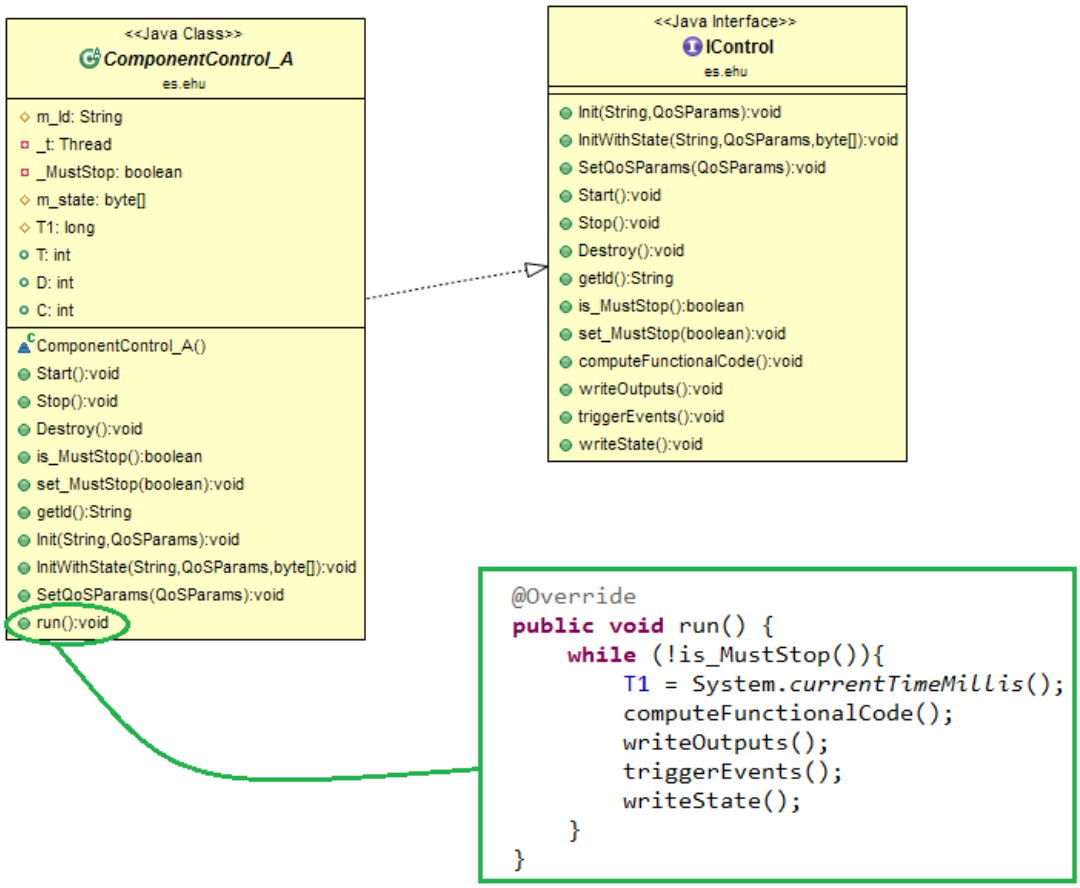


Figura 4-9: Código base para componentes SCA activados bajo demanda

Para la generación del código de una implementación de componente, el módulo Generador de Código debe:

(1) Crear una *clase Java* para el componente SCA que extiende alguna de las dos plantillas anteriores, sobrescribiendo los métodos necesarios.

(2) Generar un *composite SCA* con un único componente SCA.

(3) Crear una *Interface Java* para la recepción de datos.

Por lo tanto, son necesarios tres procesos de generación de código (*Comp*, *Composite* e *Interface*, respectivamente), con sus respectivas plantillas. En la Tabla 4-8 se relaciona cada regla de transformación de la Tabla 4-6 con la(s) plantilla(s) Acceleo desarrollada(s) y se describe el cometido de dicha(s) plantilla(s). Nótese que no todas las reglas llevan tres plantillas asociadas.

Tabla 4-8: Implementación de las reglas de transformación para DAMP con Acceleo

REGLA	PLANTILLA ACCELEO	DESCRIPCIÓN
Inicialización	Inicio_Comp	Sobrescribir el método *Init* o *InitWithState* de la clase base.
Conector de Entrada	ConEnt_Comp	Implementar el método correspondiente del Interface Java.
	ConEnt_Interface	Añadir al Interface Java un método para la recepción de las entradas.
Puerto de Entrada	P_Ent_Comp	Interpretar la lógica asociada, si fuera necesario.
	P_Ent_Composite	Añadir un servicio al composite SCA.
	P_Ent_Interface	Crear un Interface Java para la recepción de datos.
Unidad de Servicio	US_Comp	Sobrescribir el método *computeFunctionalCode* de la clase base, desde el que se invoca la ejecución de la implementación de la unidad de servicio y se recogen sus parámetros de salida.
Conector de Salida	ConSal_Comp	Añadir referencia al Interface Java del componente siguiente. Usar la referencia para el envío de datos.
	ConSal_Composite	Añadir una referencia al composite SCA.
Puerto de Salida	P_Sal_Comp	Sobrescribir el método *writeOutputs* de la clase base, en el que se define el envío de datos. Interpretar la lógica asociada, si fuera necesario.
Acción	Accion_Comp	Invocar al correspondiente servicio de la plataforma, expuesto en forma de servicio web SOAP.
Puerto de Eventos	P_Ev_Comp	Sobrescribir el método *triggerEvents* de la clase base.
Estado de Ejecución	Estado_Comp	Asignar un valor a la variable *m_state* de la clase base.
QoS Flexible	QoSFlex_Comp	Sobrescribir el método *SetQoSParams* de la clase base.
Finalización	Fin_Comp	Sobrescribir el método *Stop* de la clase base.

A modo de ejemplo, la Figura 4-10 presenta un fragmento de las plantillas Acceleo para la generación de los diferentes ficheros y en concreto, para la generación de la Interface Java para la recepción de datos.

```
[comment encoding = UTF-8 /]
[module componente('http://www.vgaa.org')/]
[comment templates/]
[import es::ehu::cadamToolSuite::generadorDAMP::templates::Composite/]
[import es::ehu::cadamToolSuite::generadorDAMP::templates::ComponenteSCA/]
[import es::ehu::cadamToolSuite::generadorDAMP::templates::P_Ent_Interface/]
[comment queries/]
[import es::ehu::cadamToolSuite::generadorDAMP::queries::qComponente/]

[template public componente(aComponente : Componente, nombreEscenario : String, nombreClase : String)]
[comment @main /]
[for (aImplementacion : Implementacion_C | aComponente.implementadoPor)]
    [comment composite SCA => Composite.mtl/]
    [file (aImplementacion.nombre.concat('.composite'), false, 'UTF-8')]
        [aImplementacion.generateComposite(nombreEscenario, aComponente.nombre, nombreClase)/]
    [/file]

    [comment componente SCA => ComponenteSCA.mtl/]
    [file (aImplementacion.nombre.concat('.composite'), false, 'UTF-8')]
        [aImplementacion.generateCompSCA (nombreEscenario, aComponente.nombre, nombreClase)/]
    [/file]

    [comment interface Java para conector de entrada => InterfaceJava.mtl/]
    [if (aImplementacion.eContainer(Componente).tienePuertoEntrada())]
        [aImplementacion.generateCompInterface(nombreEscenario, aComponente.nombre)/]
    [/if]
[/for]
[/temp
```

```
[comment encoding = UTF-8 /]
[module P_Ent_Interface('http://www.vgaa.org')/]

[comment templates/]
[import es::ehu::cadamToolSuite::generadorDAMP::templates::ConEnt_Interface/]
[comment queries/]
[import es::ehu::cadamToolSuite::generadorDAMP::queries::qComponente/]
[import es::ehu::cadamToolSuite::generadorDAMP::queries::qImplementacionComp/]

[template public generateCompInterface(aImplC : Implementacion_C, escenarioNombre : String, compNombre : String)]
[file ('I'.concat(aImplC.nombre + '.java'), false, 'UTF-8')]
    package es.ehu.generatedAgents.scn[escenarioNombre/].app[aImplC.eContainer(AplicacionAtomica).nombre/].comp[c
    import org.osoa.sca.annotations.Remotable;
    [for (aImport : String | aImplC.getImportsConexiones())]
    import [aImport/];
    [/for]

    @Remotable
    public interface I[aImplC.nombre/] {
        [for (aConectorEntrada : ConectorDatos | aImplC.eContainer(Componente).getConectoresEntrada())]
            [aConectorEntrada.generateConectorEntradaI ()/]
        [/for]
    }
[/file]
[/template]
```

Figura 4-10: Fragmento de plantillas Acceleo para DAMP

4.4.2.2 Generación de Código para la Plataforma de Gestión MAS-RECON

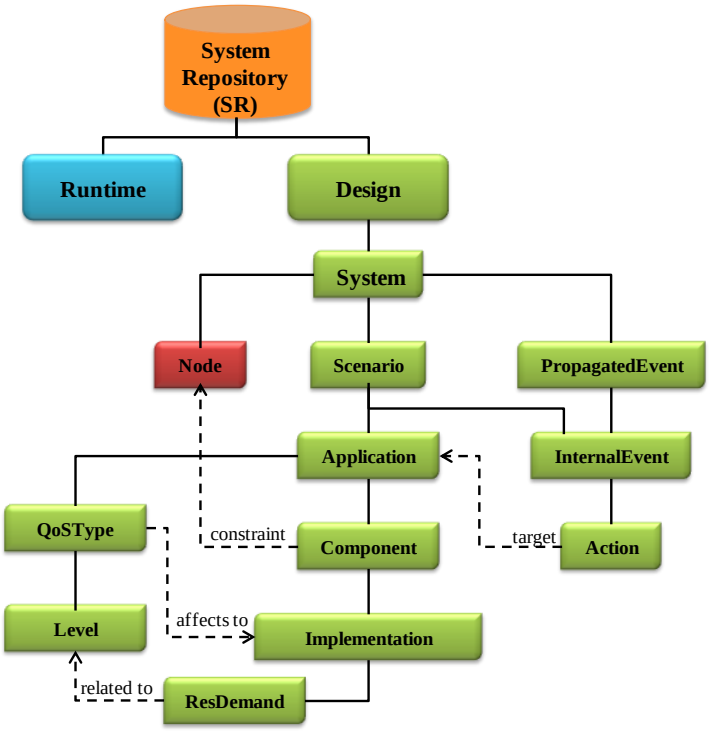
La plataforma de gestión MAS-RECON (**M**ulti**A**gent **S**ystem based management platform for dynamically **RECON**figurable systems) se basa en el framework JADE (Bellifemine et al., 2008) que facilita el desarrollo y gestión de sistemas multiagente, añadiendo nuevos módulos para hacer frente a los requisitos de las aplicaciones distribuidas sensibles al contexto (Gangoiti et al., 2016).

En la parte izquierda de la Figura 4-11 se muestra la estructura del repositorio gestionado por MAS-RECON, donde se almacena información relativa tanto al diseño como a la ejecución de todo el sistema. Únicamente se deben registrar aquellos elementos de diseño señalados en color verde. Para ello, MAS-RECON proporciona una función con la siguiente declaración:

```
reg elemento [ID=elementoID] [parent=elementoPadre]
                    listaAtributos
```

donde *parent* hace referencia al elemento de nivel superior en el repositorio y *listaAtributos* es un conjunto de parejas del tipo "nombreAtributo=valor". Nótese que la plataforma MAS-RECON permite proponer un identificador de elemento, que será aceptado si cumple la condición de unicidad dentro de su categoría.

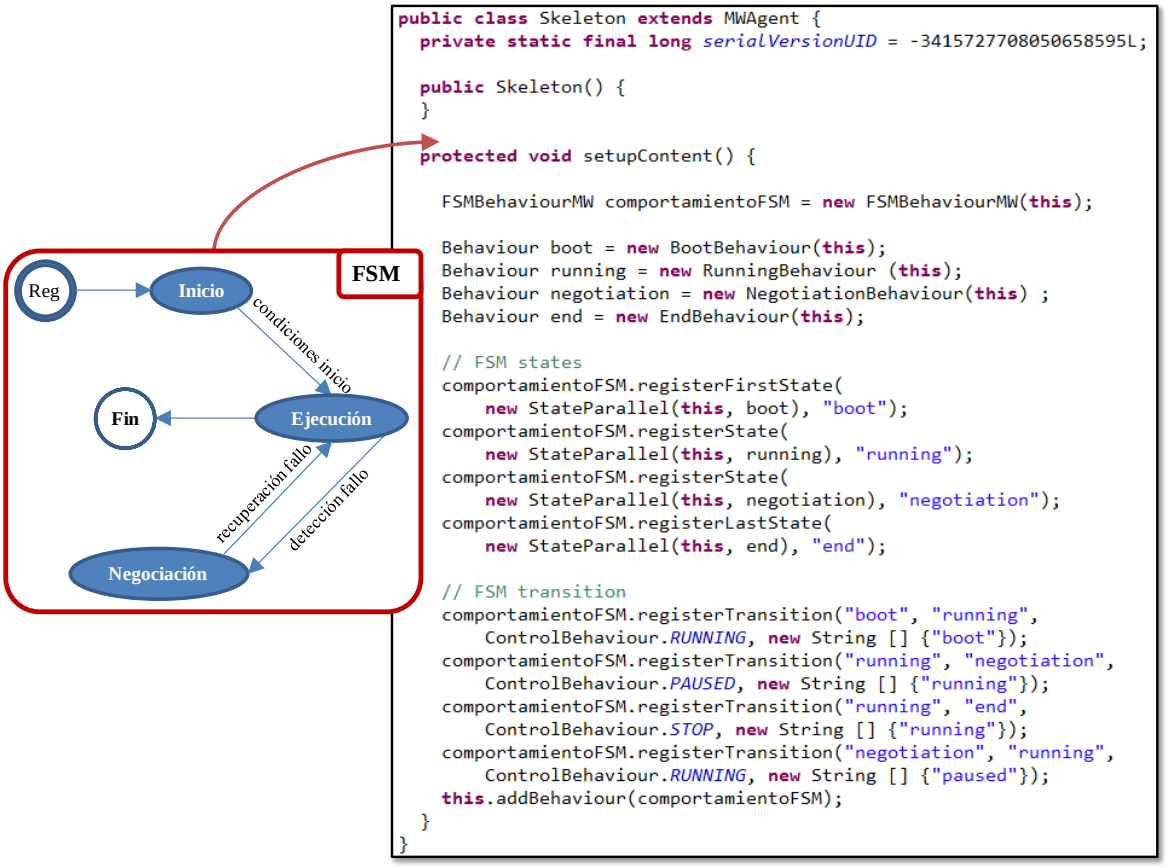
En la Figura 4-11 también se muestra la relación entre los elementos de la aproximación CADAMM y los del repositorio MAS-RECON. A diferencia de la plataforma de gestión anterior, cabe destacar que MAS-RECON proporciona mecanismos que controlan la ejecución de todas las acciones asociadas a eventos lanzados., por lo que es necesario registrar esta información.



ELEMENTO SR	ELEMENTO CADAMM
System	Sistema
Scenario	Escenario
Application	AplicaciónAtómica
Component	Componente
Implementation	Implementación_C
PropagatedEvent	TxEventoEscn
InternalEvent	Evento, RxEventoEscn
Action	Acción
QoSType	GrupoQoS (sólo si es flexible)
Level	NivelQoS
ResDemand	DemandaRecursos

Figura 4-11: Estructura del repositorio de la plataforma MAS-RECON y su relación con la aproximación de modelado CADAMM

Al igual que en la plataforma DAMP, para la implementación de los agentes MAS-RECON propone dos *plantillas* en función del tipo de activación del componente: (1) Periódica, para implementaciones de componentes que se activan periódicamente y (2) Bajo Demanda, para implementaciones de componentes que se activan tras la recepción de sus parámetros de entrada. Ambas plantillas comparten un mismo esqueleto de código (ver parte derecha de la Figura 4-12) que implementa la máquina de estados finitos (*Finite State Machine*, FSM) mostrada en la parte izquierda de la Figura 4-12. Durante el estado FSM *Inicio*, el agente espera a que se cumplan las condiciones para su inicio, lo que incluye las acciones de inicialización requeridas. Durante el estado FSM *Ejecución* el agente ejecuta su funcionalidad, actualizando su estado de ejecución en cada ciclo. Cuando se detecta un fallo, el agente pasa al estado FSM *Negociación* durante el cual espera a la restauración de su ejecución. Por último, en el estado FSM *Fin* se llevan a cabo todas las tareas de finalización necesarias.




```
public class Skeleton extends MWAgent {
  private static final long serialVersionUID = -3415727708050658595L;

  public Skeleton() {
  }

  protected void setupContent() {

    FSMBehaviourMW comportamientoFSM = new FSMBehaviourMW(this);

    Behaviour boot = new BootBehaviour(this);
    Behaviour running = new RunningBehaviour (this);
    Behaviour negotiation = new NegotiationBehaviour(this) ;
    Behaviour end = new EndBehaviour(this);

    // FSM states
    comportamientoFSM.registerFirstState(
        new StateParallel(this, boot), "boot");
    comportamientoFSM.registerState(
        new StateParallel(this, running), "running");
    comportamientoFSM.registerState(
        new StateParallel(this, negotiation), "negotiation");
    comportamientoFSM.registerLastState(
        new StateParallel(this, end), "end");

    // FSM transition
    comportamientoFSM.registerTransition("boot", "running",
        ControlBehaviour.RUNNING, new String [] {"boot"});
    comportamientoFSM.registerTransition("running", "negotiation",
        ControlBehaviour.PAUSED, new String [] {"running"});
    comportamientoFSM.registerTransition("running", "end",
        ControlBehaviour.STOP, new String [] {"running"});
    comportamientoFSM.registerTransition("negotiation", "running",
        ControlBehaviour.RUNNING, new String [] {"paused"});
    this.addBehaviour(comportamientoFSM);
  }
}
```

Figura 4-12: Máquina finita de estados y su implementación en Java, para los agentes gestionados por MAS-RECON

El módulo Generador de Código debe crear una clase Java con el *esqueleto del código de la FSM* (parte derecha de la Figura 4-12. No son necesarias plantillas Acceleo), así como una clase Java por cada uno de los *estados FSM*. Por último es necesaria una clase Java por cada *conector de datos*. Para ello se desarrolla el conjunto de plantillas Acceleo recogidas en la Tabla 4-9. Nótese que en este caso no es necesaria la plantilla para la regla Acción, ya que la gestión de las acciones asociadas a un evento es responsabilidad total de la plataforma MAS-RECON, por lo que no requiere más generación de código que la relacionada con el lanzamiento del evento.

Tabla 4-9: Implementación de las reglas de transformación para MAS-RECON con Acceleo

REGLA	PLANTILLA ACCELEO	DESCRIPCIÓN
Inicialización	Inicio	Generar la clase correspondiente al estado FSM *Inicio*.
Conector de Entrada	ConEnt_Conector	Crear una clase Java para el mensaje de datos recibido del componente anterior.
	ConEnt_Ejec	Extraer parámetros de entrada para la implementación de unidad de servicio, en la clase Java del estado FSM *Ejecución*.
Puerto de Entrada	P_Ent_Ejec	Interpretar la lógica asociada, si fuera necesario, en la clase Java del estado FSM *Ejecución*.
Unidad de Servicio	US_Ejec	Invocar la ejecución de la implementación de la unidad de servicio y recoger sus parámetros de salida, en la clase Java del estado FSM *Ejecución*.
Conector de Salida	ConSal_Conector	Crear una clase Java para el mensaje de datos enviado al siguiente componente.
	ConSal_Ejec	Construir el mensaje de datos enviado al siguiente componente, en la clase Java del estado FSM *Ejecución*.
Puerto de Salida	P_Sal_Ejec	Interpretar la lógica asociada, si fuera necesario, en la clase Java del estado FSM *Ejecución*.
Puerto de Eventos	P_Ev_Ejec	Interpretar la lógica asociada, en la clase Java del estado FSM *Ejecución*.
Estado de Ejecución	Estado_Ejec	Construir el mensaje de actualización del estado de ejecución para la plataforma de gestión, en la clase Java del estado FSM *Ejecución*.
	Estado_Neg	Establecer el estado de ejecución indicado por la plataforma de gestión, en la clase Java del estado FSM *Negociación*.
QoS Flexible	QoSFlex_Ejec	Implementar el correspondiente comando de control de la plataforma MAS-RECON correspondiente, en la clase Java del estado FSM *Ejecución*.
Finalización	Fin	generar la clase correspondiente al estado FSM Fin.

La Figura 4-13 muestra un fragmento de una de las plantillas Acceleo implementadas para la generación del código. En concreto, la correspondiente a la personalización del esqueleto del agente.

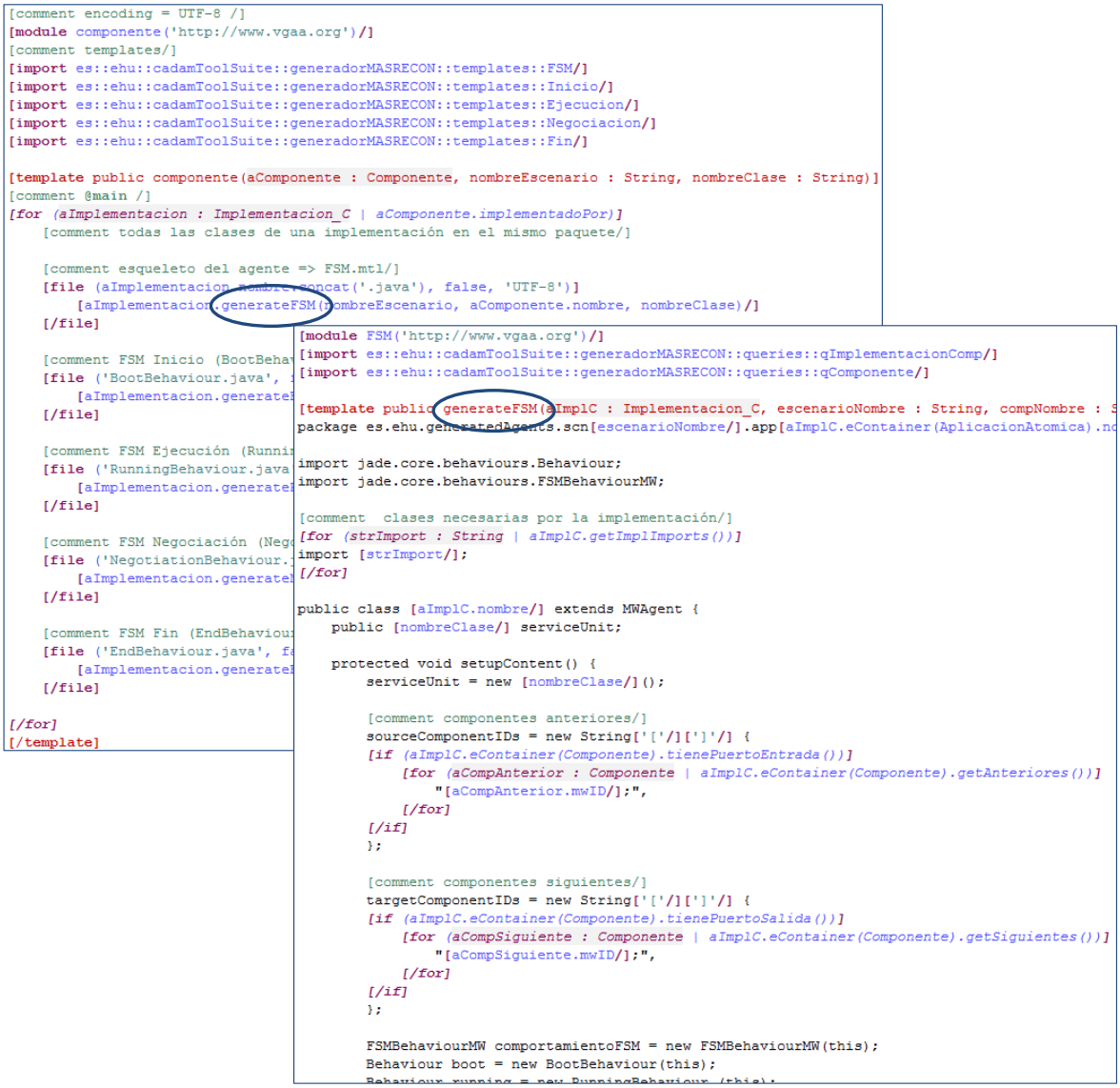


Figura 4-13: Fragmento de plantillas Acceleo para MAS-RECON

4.5 Gestor de Ejecución

Este módulo **encapsula la interacción** con la plataforma de gestión en fase de ejecución, **abstrayendo** a los usuarios de CADAMToolSuite de las particularidades de su implementación. Dicha interacción se refiere a dos situaciones: (1) el control de la

ejecución de las aplicaciones y (2) la monitorización de las aplicaciones en ejecución y de los recursos del sistema.

Se trata de un módulo totalmente dependiente de la plataforma de gestión ya que incluye llamadas a su API. No obstante, con objeto de que el Editor Gráfico pueda acceder a cualquier plataforma de forma transparente, se ha definido una interfaz común, cuya implementación en forma de plug-in contiene las particularidades de una plataforma de gestión concreta.

4.5.1 Arranque / Parada

Uno de los objetivos del *Gestor de Ejecución* es permitir que los usuarios puedan solicitar a la plataforma de gestión el **inicio o parada de la ejecución** de determinadas aplicaciones. La **API** ofrecida por el Gestor de Ejecución al Editor Gráfico para ello se muestra en la Tabla 4-10.

Resulta necesario señalar que únicamente se pueden arrancar aquellas aplicaciones que hayan sido previamente registradas, y que además no sean activadas por evento o pertenezcan a una súper-aplicación. Con respecto a la finalización de su ejecución, cuando se solicita la detención de una aplicación también se detienen todas aquéllas con las que está relacionada a través de acciones asociadas a eventos lanzados por sus componentes. Por lo tanto, no se puede solicitar la parada de aquéllas que se detienen por evento. En ese caso se debe solicitar la detención de la aplicación que lanza el evento cuya acción asociada la detiene. Tampoco se puede solicitar la finalización de aplicaciones que pertenecen a una súper-aplicación. Finalmente, destacar que en cualquier caso, es la plataforma de gestión la que elige el instante concreto para detener las aplicaciones.

Tabla 4-10: API del Gestor de Ejecución para arranque/parada

MÉTODO	DESCRIPCIÓN
arrancarAplicAtomica	Solicita la activación de la aplicación seleccionada, si no es activada por evento y si no forma parte de una súper-aplicación. `void arrancarAplicAtomica()`
arrancarSuperAplic	Solicita el arranque de todas las aplicaciones atómicas de la súper-aplicación seleccionada, si ésta no es activada por evento. `void arrancarSuperAplic()`
arrancarEscenario	Solicita el arranque de todas las súper-aplicaciones y aplicaciones atómicas del escenario seleccionado, no activadas por evento. `void arrancarEscenario()`
detenerAplicAtomica	Solicita la finalización de la ejecución de la aplicación atómica seleccionada, siempre que no forme parte de una súper-aplicación y no sea detenida por evento. También se detienen todas sus aplicaciones relacionadas mediante eventos. `void detenerAplicAtomica()`
detenerSuperAplic	Solicita la finalización de todas las aplicaciones atómicas de la súper-aplicación seleccionada, si no es detenida por evento. `void detenerSuperAplic()`
detenerEscenario	Solicita la finalización de todas las súper-aplicaciones y aplicaciones atómicas del escenario seleccionado. `void detenerAplicAtomica()`

4.5.2 Monitorización en Tiempo de Ejecución

Para la monitorización en tiempo de ejecución por parte de un operador, en este apartado se propone una solución que permita conocer el **estado del sistema en un instante de ejecución** concreto así como el **histórico de eventos de plataforma** ocurridos, capturados en el llamado **modelo de ejecución**. Se trata, por lo tanto, de información que únicamente puede proporcionar la plataforma de gestión, pero que debe ser mostrada al operario en un formato amigable e independiente de plataforma.

En este contexto, el diseño de esta parte del módulo Gestor de Ejecución persigue dos objetivos fundamentales. Por un lado, ofrecer un *API común* al Editor Gráfico. Por otro lado, definir un *formato de presentación único* para el modelo de ejecución, independientemente de cuál sea la plataforma de gestión que lo proporcione.

Con respecto a la **API** ofrecida al Editor Gráfico, se definen tres métodos recopilados en la Tabla 4-11.

Tabla 4-11: API del Gestor de Ejecución para monitorización en tiempo de ejecución

MÉTODO	DESCRIPCIÓN
solicitarHistorico	Solicita información relativa a los eventos de plataforma ocurridos desde un instante de tiempo indicado. `Document solicitarHistorico (Date instante)`
solicitarEstado	Solicita información relativa al estado de ejecución del sistema en el instante actual. `Document solicitarEstado()`
solicitarMEjecucion	Solicita el modelo de ejecución completo con información relativa a los eventos de plataforma desde un instante indicado y al estado de ejecución actual del sistema. `Document solicitarMEjecucion (Date instante)`

Cada uno de estos métodos transforma la información proporcionada por la plataforma de gestión a la estructura de datos mostrada en la Figura 4-14, almacenando el modelo obtenido en el Repositorio de Modelos. Por cada **evento de plataforma** se debe conocer: el *instante* en el que se ha producido, teniendo en cuenta que todos los equipos están sincronizados mediante el protocolo *Network Time Protocol* (NTP) (Network Time Foundation, 2016), el *tipo* de evento, y si fuera necesario, el elemento relacionado con dicho evento así como información adicional sobre lo ocurrido. En concreto, toda la información que la plataforma de gestión proporciona sobre un evento, que no tiene representación en la estructura de datos propuesta, puede guardarse en la propiedad *infoAdicional*. De esta manera, se puede tener conocimiento de situaciones como por ejemplo si tras la sobrecarga de un nodo

ha sido necesario modificar el nivel de calidad de servicio de alguna aplicación o el lanzamiento de un evento por parte de una aplicación.

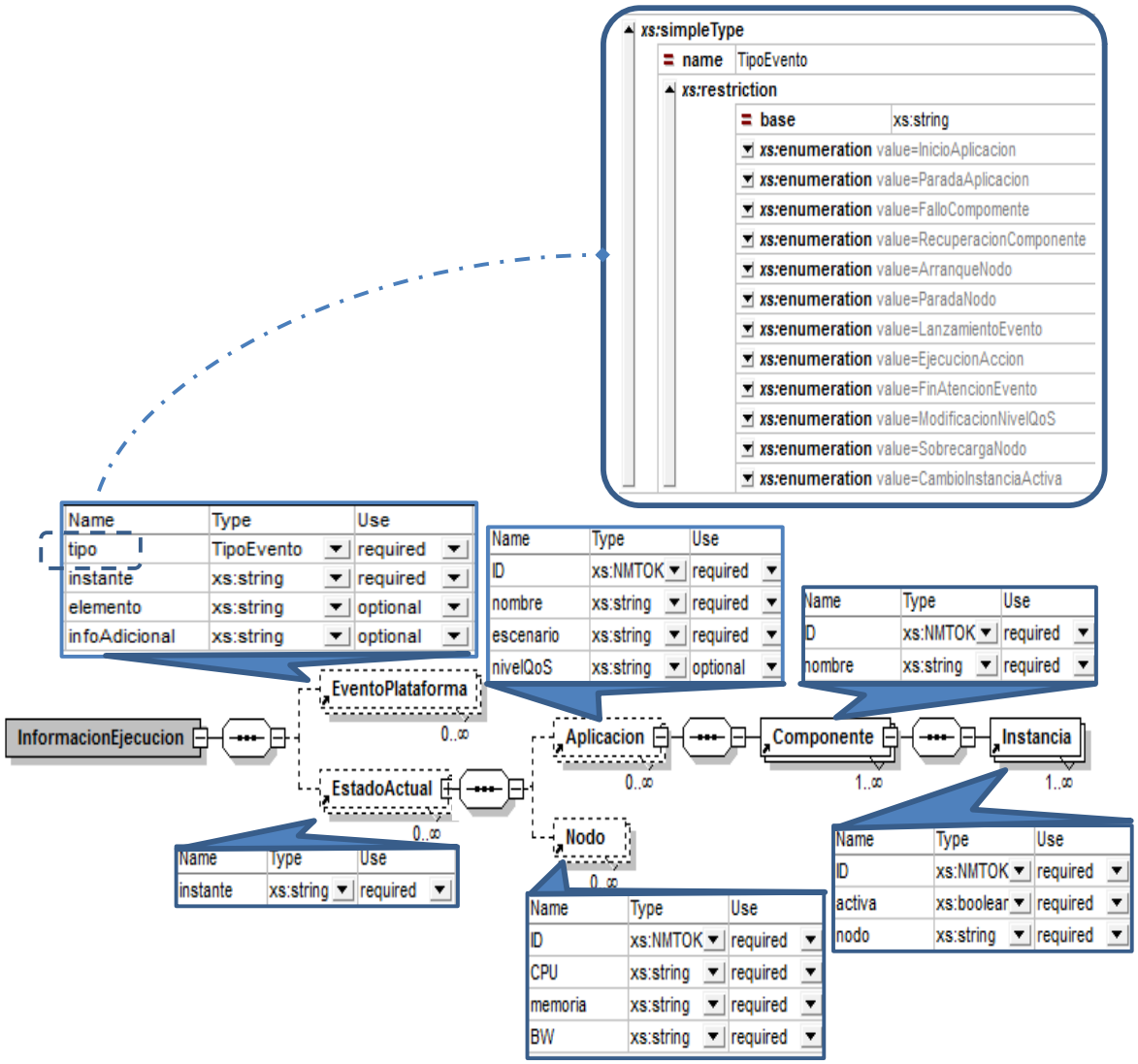


Figura 4-14: Estructura del modelo de ejecución

Con respecto al **estado actual** del sistema en un *instante* concreto, se proporciona información acerca de los nodos y aplicaciones activas. El estado de un **nodo** viene determinado por el de sus recursos: carga de *CPU*, de *memoria* y de *ancho de banda*. Por cada **aplicación** activa se requiere información sobre las **instancias** de sus **componentes**, en qué *nodo* están desplegadas y cuál está *activa* (únicamente una de todas las instancias). También es interesante conocer el *nivel de QoS* en el que se está ejecutando, en caso de que soporte calidad de servicio flexible.

4.6 Conclusiones

En este capítulo se ha presentado CADAMToolSuite, un entorno integrado por cuatro módulos que da soporte al ciclo de desarrollo de las aplicaciones de interés: el Editor Gráfico, el Generador de Código, el Gestor de Ejecución y el Repositorio de Modelos. Se ha propuesto un diseño modular en el que cada módulo es **independiente** de la **implementación concreta** del resto, estableciéndose las relaciones entre ellos en base a un **API** establecida, que se implementa en forma de plug-in de Eclipse. Además la **colaboración** entre los diferentes módulos de la herramienta se basa en el **modelo de aplicaciones** especificado.

De hecho, el Editor gráfico propuesto permite que el **experto** de dominio realice el **diseño arquitectónico** de las aplicaciones mediante una *interfaz gráfica y amigable* que **guía** dicha especificación, la cual además es totalmente independiente de la plataforma. Los mecanismos ofrecidos por el Editor Gráfico permiten asegurar la construcción de un **modelo** de aplicaciones **correcto**, de tal manera que el experto no es consciente de que en realidad está trabajando con técnicas de modelado. Por su parte, el diseño del **desarrollador de software** se centra en la caracterización y desarrollo de las unidades mínimas de funcionalidad, las *unidades de servicio y sus implementaciones*, y almacenarlas en el Repositorio de Modelos. Así, el Editor Gráfico y el modelo de aplicaciones son el nexo de unión entre los dos expertos, permitiendo su **colaboración**. Es más, gracias al almacenamiento de las unidades de servicio en el Repositorio de Modelos, es posible definir una aplicación en base a una *composición de unidades de servicio* previamente desarrolladas, encapsuladas en componentes.

Por otro lado, los módulos Generador de Código y Gestor de Ejecución permiten **abstraer** a los usuarios de CADAMToolSuite de las **particularidades de la plataforma de gestión**. El uso de **transformaciones M2T**, en forma de plantillas Acceleo, permite automatizar el proceso de **generación de código sin errores**. Esto es posible ya que por un lado parte de un diseño correcto y por otro lado se evitan errores humanos derivados de tareas repetitivas. Además, este módulo también automatiza el proceso de registro en la plataforma de gestión.

En fase de ejecución, el Gestor de Ejecución facilita la **monitorización** de las aplicaciones y los recursos del sistema, de forma transparente a la plataforma de gestión. Se ha definido un *formato único e independiente de plataforma* de manera que el operador puede acceder a dicha información siempre de la misma manera.

En resumen, se puede concluir que el entorno CADAMToolSuite, basado en la aproximación de modelado CADAM, **cubre el ciclo de desarrollo** de las aplicaciones objeto de este trabajo de investigación: desde su especificación hasta su ejecución, pasando por su generación de código.

5 CASO DE ESTUDIO: ASISTENCIA DOMICILIARIA

"La tecnología por sí sola no basta.
También tenemos que poner el corazón."

(Hellen Keller)

5.1 Introducción

Para probar la validez tanto de la aproximación de modelado **CADAMM** como del entorno **CADAMToolSuite** se ha puesto en marcha un demostrador en el campo de aplicación de **asistencia domiciliaria**, en el que **MAS-RECON** se emplea como plataforma de gestión.

Más concretamente, el demostrador se centra en una residencia de ancianos. En el primer apartado de este capítulo se plantean las diferentes necesidades de la residencia a desarrollar. A lo largo de los siguientes apartados se presentan la especificación y desarrollo de dicho demostrador, para lo cual se hace uso de CADAMToolSuite, demostrando así que:

(1) la aproximación de modelado CADAMM permite capturar toda la información necesaria para la generación de código y para que la plataforma MAS-RECON pueda cumplir con los requisitos de las aplicaciones en ejecución.

(2) el entorno CADAMToolSuite soporta el ciclo de desarrollo de las aplicaciones.

5.2 Residencia de Ancianos

La asistencia domiciliaria permite hacer frente al alto coste que el incesante envejecimiento de la población supone para los sistemas de salud públicos. Como se ha comentado en el Capítulo 1, las llamadas casas inteligentes aumentan el abanico de posibilidades de la asistencia domiciliaria, tradicionalmente orientada al intercambio de información entre paciente y médico, o a la gestión de alarmas solicitadas por el propio paciente. Como resultado, se plantean nuevos escenarios como los descritos en este apartado en relación a una residencia de ancianos e ilustrados en trabajos de otros autores como por ejemplo (Chen, 2011; Holborn et al., 2003; Jobbágy et al., 2006; Hervás et al., 2013).

El demostrador desarrollado pretende emular una residencia de una sola planta con tres habitaciones y una zona común. En la residencia conviven tres ancianos, uno en

cada habitación, y un equipo médico formado por dos personas. Para cada residente se definen los siguientes controles de salud:

- *Residente 1*: no presenta ningún problema de salud serio, aunque ha sufrido una intervención quirúrgica. Por lo tanto, su temperatura debe controlarse cuatro veces al día. Los datos medidos se almacenan para análisis posteriores, y en caso de que alguno supere un determinado umbral se avisará al personal médico, quien supervisará si se trata de una infección.

- *Residente 2*: sufre de tensión alta, por lo que se debe supervisar su tensión arterial cuatro veces al día, siguiendo el procedimiento descrito en (Jobbágy et al., 2006). Así, antes de medir la tensión arterial se supervisará el pulso del paciente (una lectura cada 30 segundos), hasta que se alcance un tiempo de espera máximo (4 minutos) o hasta que se relaje, momento en el que se iniciará la lectura de la tensión arterial. En caso de que el pulso o la tensión estén fuera de rango, se avisará al personal médico.

- *Residente 3*: tiene problemas de corazón, por lo que su frecuencia cardiaca se controla cada 10 minutos. Si el pulso incrementa de forma anormal, además de avisar al personal médico, se debe incrementar la frecuencia de medida, lo que proporcionará información más precisa.

Por otro lado, en la *zona común* del edificio está equipada con un sistema de detección de incendios, basado en el análisis de la temperatura ambiental y la concentración de CO_2, capturadas cada minuto. En caso de fuego, se iniciará la monitorización de los signos vitales básicos de todos los residentes (pulso y concentración de oxígeno en sangre, también cada minuto). Como resultado, los servicios de emergencia dispondrán de información relativa al estado de salud de cada paciente, lo que facilitará y agilizará su atención.

También se dispone de un repositorio para almacenar información de los pacientes: datos personales (identificador, nombre, edad, habitación asignada, etc.) y datos médicos en función de su problemática (frecuencia cardiaca máxima, frecuencia cardiaca en reposo y rango normal de temperatura corporal, entre otros).

En la Tabla 3-1 se muestra cómo los requisitos de las aplicaciones distribuidas sensibles al contexto identificados en el Capítulo 3 están presentes en el demostrador de la residencia de ancianos.

Tabla 5-1: Requisitos de aplicaciones distribuidas sensibles al contexto en el demostrador de la residencia de ancianos

REQUISITO	DEMOSTRADOR
R1 Distribución de aplicaciones	La información del contexto se captura mediante sensores (biomédicos o ambientales), y su procesamiento se lleva a cabo en equipos con distintas capacidades, localizados en diferentes puntos de la residencia.
R2 Adquisición, procesamiento y actuación personalizados	Tanto la toma de medidas como su procesamiento, las acciones de actuación y los umbrales de decisión son dependientes de la problemática de cada residente.
R3 Escalabilidad	En la residencia puede haber un residente como mínimo y tres como máximo. Además, los dispositivos de adquisición o actuación, así como los de procesamiento, pueden variar a lo largo del tiempo.
R4 Calidad de servicio específica de aplicación	Cada actividad de supervisión presenta diferentes requisitos temporales.
R5 Calidad de servicio flexible	Las situaciones descritas no soportan degradación de su QoS.
R6 Tipos de activación	Existen actividades de supervisión activadas por tiempo, como la medida de la temperatura del Residente_1 o la monitorización de la frecuencia cardiaca del Residente_3. Sin embargo, la medida de la tensión arterial del Residente_2 se activa por evento (relajación del residente), siendo además de ejecución única.
R7 Adaptabilidad	Excepto en el caso del Residente_1, se pueden encontrar ejemplos de cambios de contexto frente a los que reaccionar: relajación del Residente_2 (detener la monitorización del pulso e iniciar la medida de la tensión arterial), alta frecuencia cardiaca del Residente_2 (aumentar la frecuencia de adquisición) y detección de incendio en la zona común (activación de monitorización de constantes vitales básicas).

REQUISITO	DEMOSTRADOR
R8 Disponibilidad	Es necesario asegurar la continuidad del servicio, o la restauración tan pronto como sea posible, para evitar la pérdida de información, sobre todo en caso de emergencia.
R9 Seguridad	Los datos sobre pacientes gestionados deben estar protegidos.

5.3 Repositorio de Modelos

En este apartado se describen los modelos que se almacenan en el repositorio, previamente a la especificación de la residencia de ancianos.

5.3.1 Modelo de Recursos

El prototipo del demostrador consta de sensores biomédicos, sensores ambientales, y equipos de procesamiento (ver Figura 5-1). En concreto, para la monitorización de constantes vitales se hace uso del llamado *e-Health Sensor Platform V2.0*. (Cooking Hacks, 2013a). Se trata de una plataforma para sensores biomédicos que se puede montar sobre Arduino o Raspberry Pi. De todos los sensores que ofrece, en este demostrador se han empleado el de temperatura corporal, el pulsioxímetro para la medición del pulso y el tensiómetro. Cada paciente dispone de una de estas plataformas, montada sobre una Raspberry Pi, con los sensores adecuados a sus problemas de salud. En cuanto a la supervisión del entorno, se realiza mediante sensores de temperatura ambiental y de CO_2 montados sobre waspmotes (Cooking Hacks, 2013b). Únicamente la zona común está provista de un waspmote de detección. Para las tareas de procesamiento se dispone de 4 PCs, mientras que el equipo médico dispone de dos Raspberry Pi con pantalla LCD gráfica.

Figura 5-1: Infraestructura del demostrador: e-Health Sensor Platform, kit de sensores de gas y unidades de procesamiento

En resumen, la infraestructura está formada por:

- Sensores:
  - 1 x sensor de temperatura corporal (montado en *e-Health Sensor Shield*)
  - 1 x tensiómetro (montado en *e-Health Sensor Shield*)
  - 3 x pulsioxímetro (montado en *e-Health Sensor Shield*)
  - 1 x sensor de temperatura ambiental (montado sobre waspmote)
  - 1 x sensor de concentración de CO_2 (montado sobre waspmote)
- Equipos de visualización:
  - 2 x pantalla LCD gráfica
- Equipos de procesamiento:
  - 5 x Raspberry Pi
  - 2 x waspmote
  - 4 x PC

Para la caracterización de estos elementos se emplea la aproximación propuesta en un trabajo anterior enmarcada dentro del proyecto iLAND (Calvo et al., 2012), y que

se tiene en cuenta para la implementación de los métodos de la API del Repositorio de Modelos.

5.3.2 Modelo del Entorno Físico

En el caso de la residencia de ancianos, este modelo representa el plano de la residencia y el equipamiento de sensores y actuadores del que está dotada. A modo de ejemplo, la Figura 5-2 muestra una representación gráfica de dicho modelo para la residencia antes descrita: una única planta con 3 habitaciones disponibles. A cada residente se le ha asignado una habitación, y existe una zona común a todos ellos.

En este contexto, el método *extraerInfoEntorno* de la API del Repositorio de Modelos se encarga de procesar este modelo para obtener las habitaciones y zonas disponibles junto con su instrumentación.

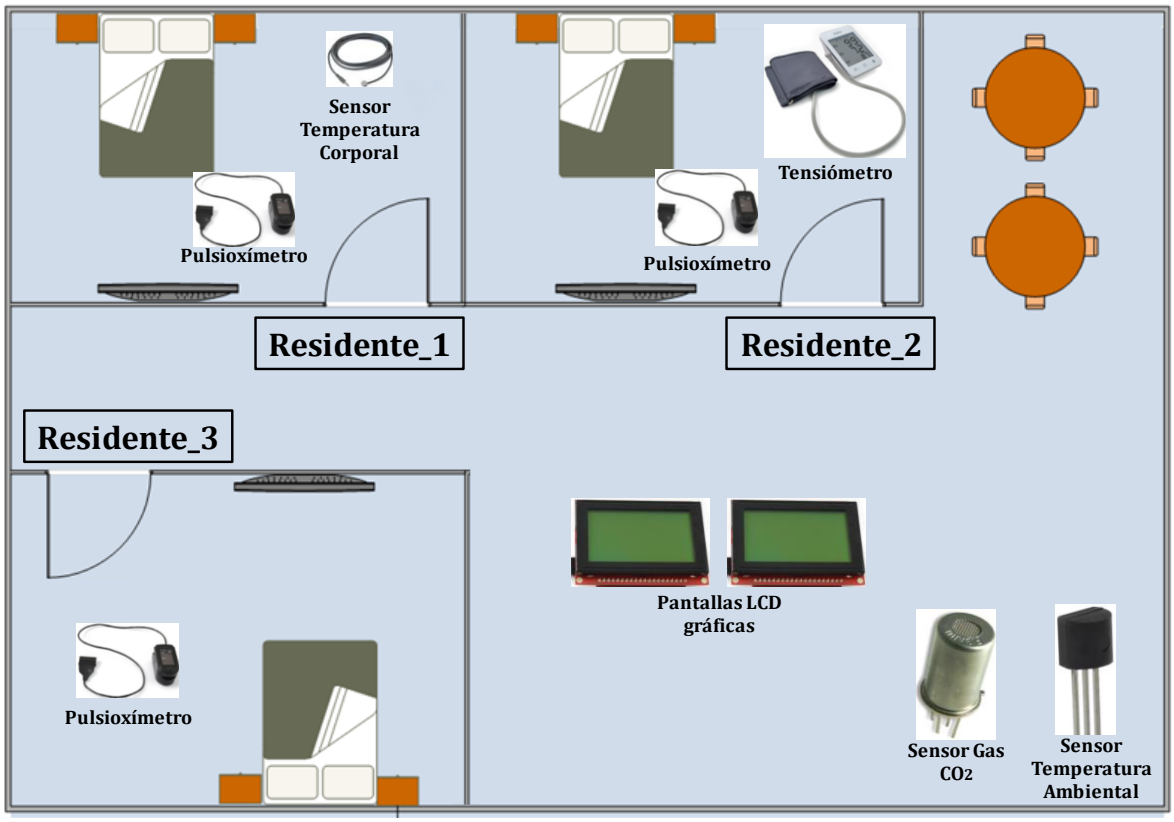


Figura 5-2: Representación gráfica del modelo del entorno físico para el demostrador Residencia de Ancianos

5.3.3 Unidades de Servicio e Implementaciones

Antes de realizar la especificación de las aplicaciones, se caracterizan todas las unidades de servicio necesarias así como sus implementaciones y se almacenan en el repositorio. Por simplicidad, se ha desarrollado una única implementación por unidad de servicio, empleando el lenguaje de programación orientado a objetos Java.

A modo de ejemplo la Figura 5-3 muestra la caracterización de la unidad de servicio *ComprobarPacienteRelajado*, encargada de determinar si un paciente se ha relajado en base al análisis de varias medidas de pulso consecutivas. Para ello, precisa de dos parámetros de entrada: pulso (*pulso*) e instante de medida (*instanteMedida*); y proporciona cuatro parámetros de salida: pulso medio (*pulsoMedio*), instante de la última medida (*instanteMedida*), indicador de relajación (*pacienteRelajado*) e indicador de tiempo límite agotado (*tiempoEsperaAgotado*). Resulta importante destacar que se ha definido una caracterización de unidad de servicio que no se corresponde únicamente con información relativa a la vista del desarrollador de SW. Así, los campos *nombre*, descripción (*desc*) y *unidades* se refieren a la caracterización del experto de dominio con respecto a los datos del componente. De esta forma se define la relación existente entre los datos de entrada/salida del componente y los parámetros de entrada/salida de la unidad de servicio.

UnidadServicio

nombre	ComprobarPacienteRelajado
desc	Comprueba si el paciente ha alcanzado el estado de relajación analizando medidas consecutivas de su pulso
inicializacion	si
finalizacion	no
conEstado	si

Entradas — ParamEntrada (2)

	nombreExp	unidades	nombre	tipoDato	desc
1	pulso	pulsaciones/minuto	iPulso	long	Pulso del paciente en pulsaciones por minuto
2	instanteMedida	día y hora (dd-mm-aaaa hh:mm:ss)	dInstante	DATE	Instante en el que se capturó la medida de pulso

Salidas — ParamSalida (4)

	nombreExp	unidades	nombre	tipoDato	desc
1	esperaAgotada	Si/No	bAgotado	boolean	Indica si el tiempo de espera a la relajación se ha agotado
2	pulsoMedio	pulsaciones/minuto	iMedia	long	Valor medio del pulso del paciente
3	instanteMedida	día y hora (dd-mm-aaaa hh:mm:ss)	dInstante	DATE	Instante de la última medida tomada
4	enRelax	Si/No	bRelax	boolean	Indica si el paciente se ha relajado

ParametrosConfiguracion — ParamConfig (2)

	nombre	tipoDato	desc
1	pacienteID	string	Identificador único del paciente
2	tiempoEspera	long	Tiempo máximo de espera a la relajación

Figura 5-3: Caracterización de la unidad de servicio ComprobarPacienteRelajado

Se trata de una unidad de servicio que requiere una configuración inicial (presenta parámetros de configuración) y el mantenimiento de su estado (precisa de medidas de pulso anteriores tanto para calcular la media como para determinar si el paciente se ha relajado). Por lo tanto, su implementación, mostrada en la Figura 5-4, dispone de los métodos *inicializar*, *procesar*, *leerEstado* y *escribirEstado*, descritos en el Capítulo 3.

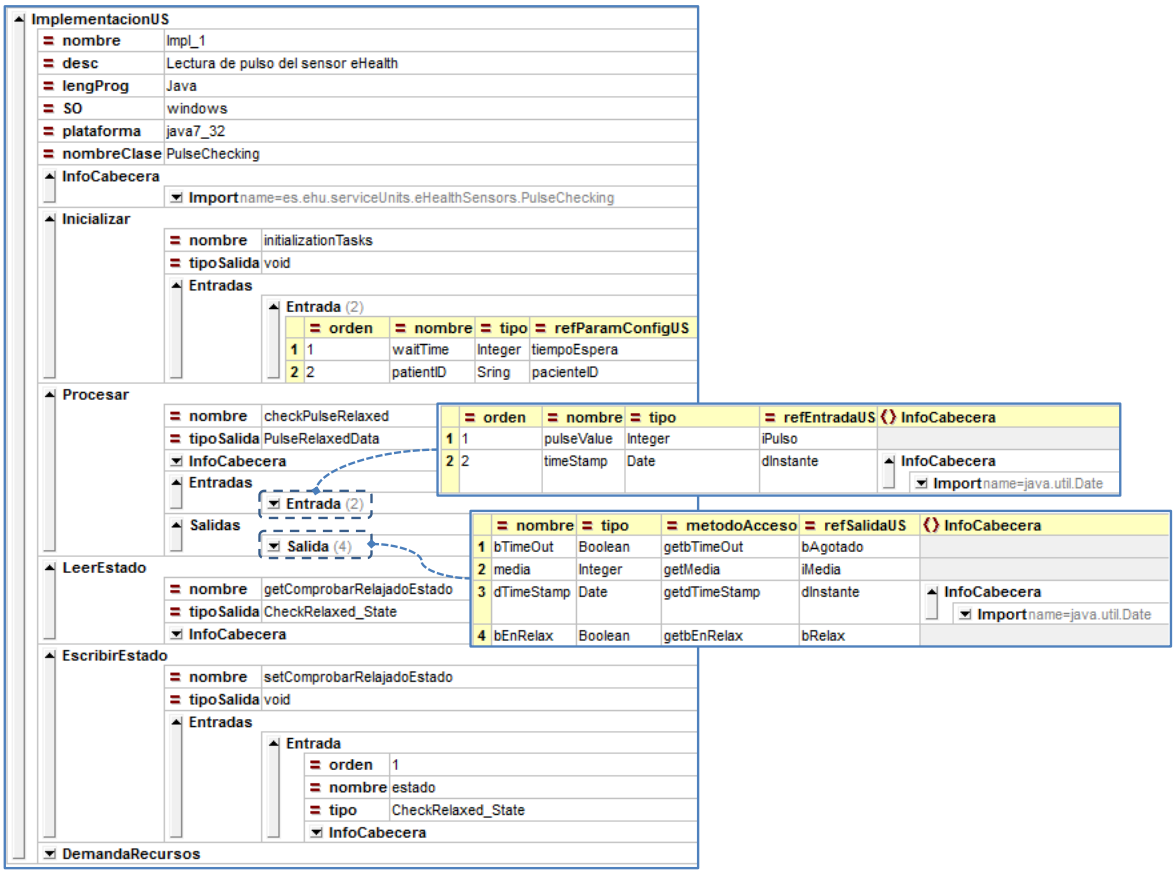


Figura 5-4: Caracterización de la implementación de la unidad de servicio ComprobarPacienteRelajado

5.4 Especificación y diseño

Como se ha comentado en el Capítulo 4, el punto de partida del editor gráfico es el meta-modelo MM_Aplicaciones, una personalización de la aproximación de modelado CADAMM para un campo de aplicación concreto. En este apartado se describen las personalizaciones necesarias para el ámbito de la asistencia domiciliaria, y a continuación se detalla cómo se ha hecho uso del Editor Gráfico para el modelado del demostrador.

5.4.1 MM_Aplicaciones

En el caso de asistencia domiciliaria, y más concretamente para el demostrador desarrollado, las personalizaciones se refieren principalmente a la definición de la

QoS, que tiene en cuenta los requisitos temporales. La Figura 5-5 muestra esta personalización, donde, por simplicidad, únicamente se muestran los elementos de modelado afectados.

En primer lugar se limitan los *Tipos de QoS* posibles a periodo y tiempo límite (tiempo máximo permitido durante el cual la aplicación puede estar activa, *timeout*). Toda aplicación debe tener como mínimo un Grupo de QoS con un Tipo de QoS para el tiempo límite. Todas las súper-aplicaciones son periódicas por lo que deben tener el Tipo de QoS correspondiente. Además, el periodo de la súper-aplicación debe ser mayor que el de sus atómicas periódicas.

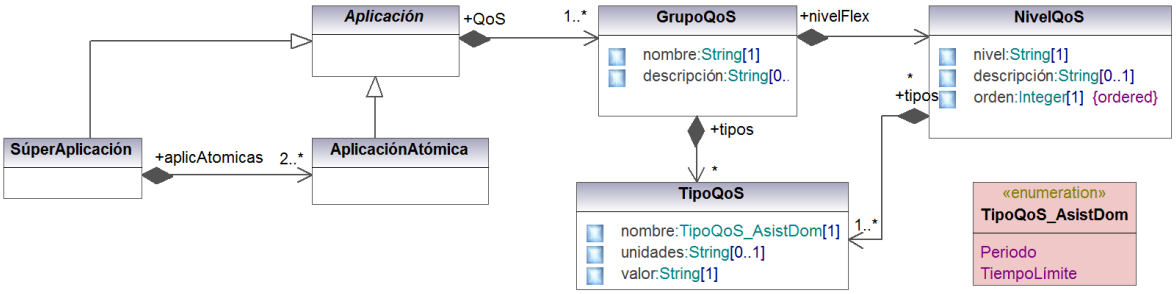


Figura 5-5: Fragmento del meta-modelo MM_Aplicaciones para asistencia domiciliaria

5.4.2 Modelado

Para la **especificación y diseño** de las aplicaciones del demostrador de la residencia de ancianos se ha hecho uso del Editor Gráfico. La Figura 5-6 muestra un fragmento del modelo M_Aplicaciones obtenido.

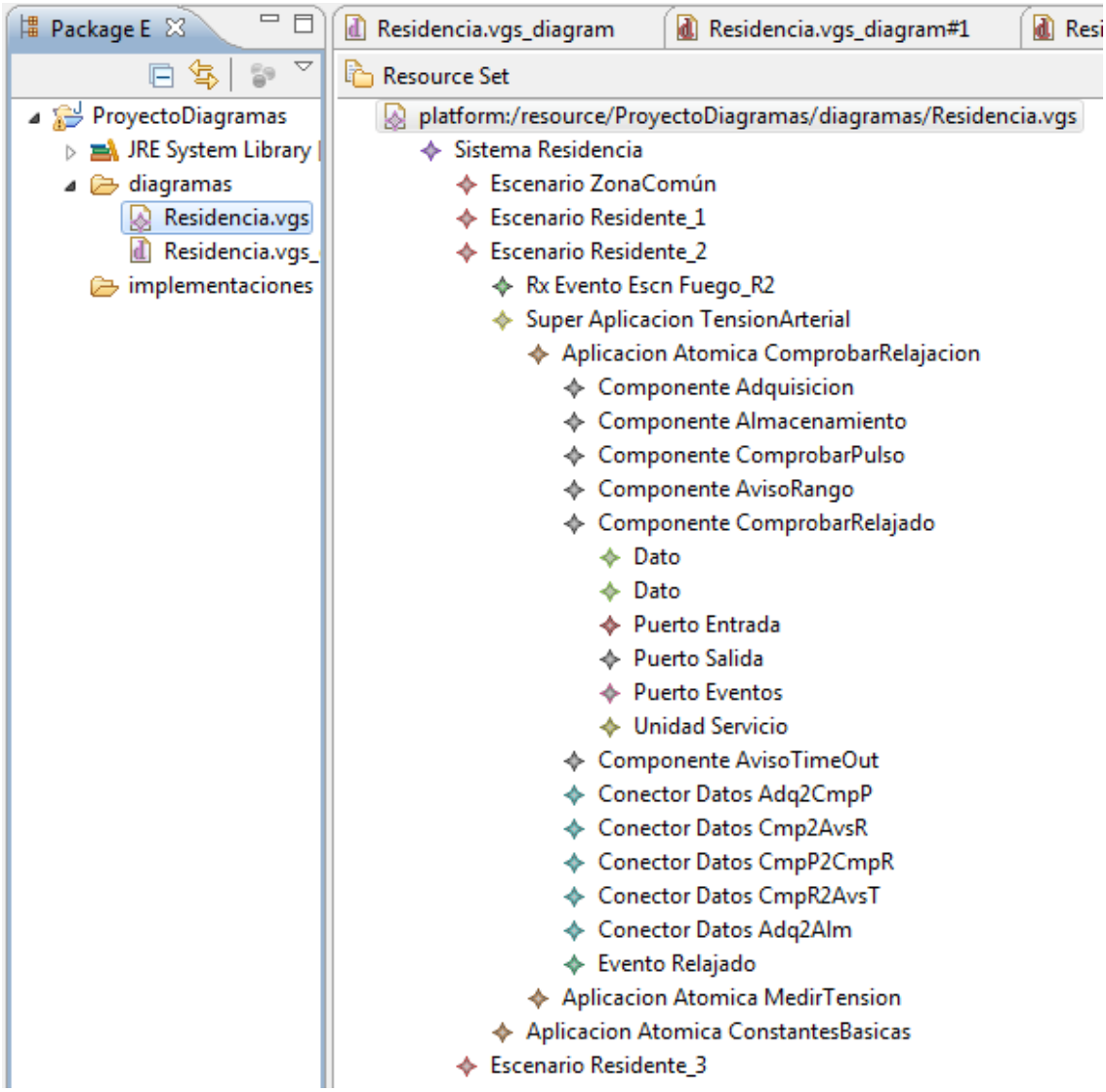


Figura 5-6: Fragmento del modelo M_Aplicaciones

A continuación se describen los pasos a seguir por el experto de dominio (los cuatro primeros), y por el desarrollador de software (el resto). Se debe tener en cuenta que, como se ha comentado en el apartado 5.3.3, se parte de una situación en la que todas las unidades de servicio necesarias están almacenadas en el Repositorio de Modelos. Lo mismo ocurre con las implementaciones de unidades de servicio.

1) Crear escenarios

En asistencia domiciliaria los escenarios agrupan todas las actividades de supervisión de la salud de un paciente. También pueden referirse a la supervisión de zonas de un edificio. Por lo tanto, partiendo del modelo del entorno físico almacenado en el

Repositorio de Modelos (ver Figura 5-2), en la Vista Gráfica del Sistema *Residencia* mostrada en la Figura 5-7 se definen cuatro escenarios: uno por cada residente (*Residente_1*, *Residente_2* y *Residente_3*) y el de la zona común (*ZonaComún*).

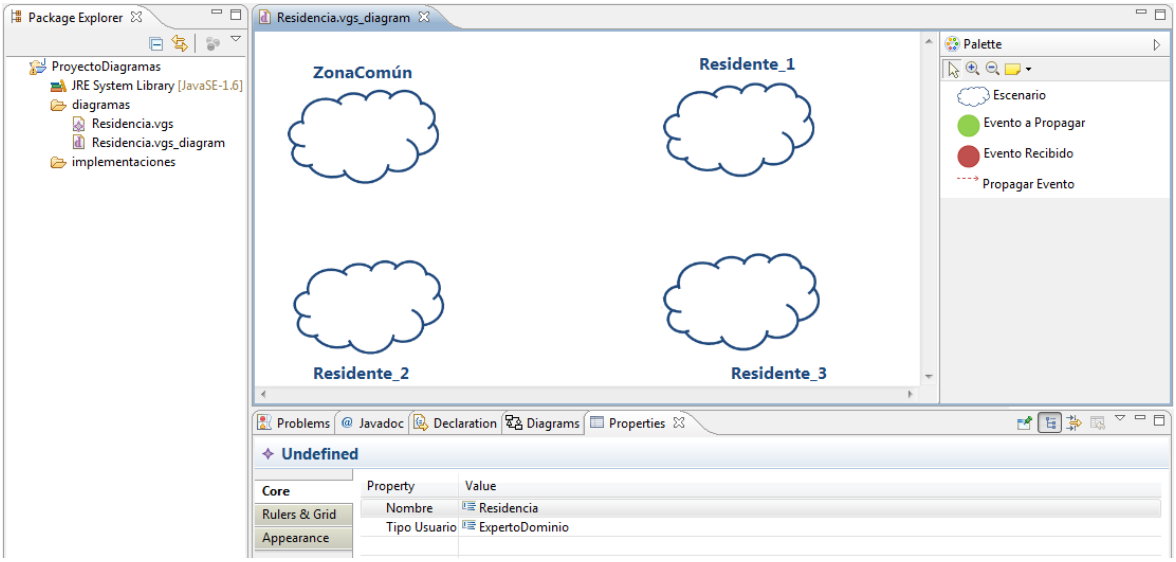


Figura 5-7: Vista Gráfica del Sistema *Residencia*. Definición de escenarios

2) Crear aplicaciones y definir su QoS

Para la especificación de cada escenario concreto se hace uso de su correspondiente Vista Gráfica de Escenario. En todos los escenarios de residentes (*Residente_1*, *Residente_2* y *Residente_3*) se ha creado una aplicación atómica llamada *ConstantesBásicas* para la monitorización de las constantes vitales básicas. Dichas aplicaciones tienen una QoS relativa a sus requisitos temporales (grupo de QoS), con un tiempo límite de 1 minuto (tipos de QoS).

Como se muestra en la Figura 5-8, en el escenario *Residente_1* también se ha definido una aplicación atómica (*TempCorporal*) para la monitorización de su temperatura corporal. Su periodo es de 6 horas y su tiempo límite de 15 minutos.

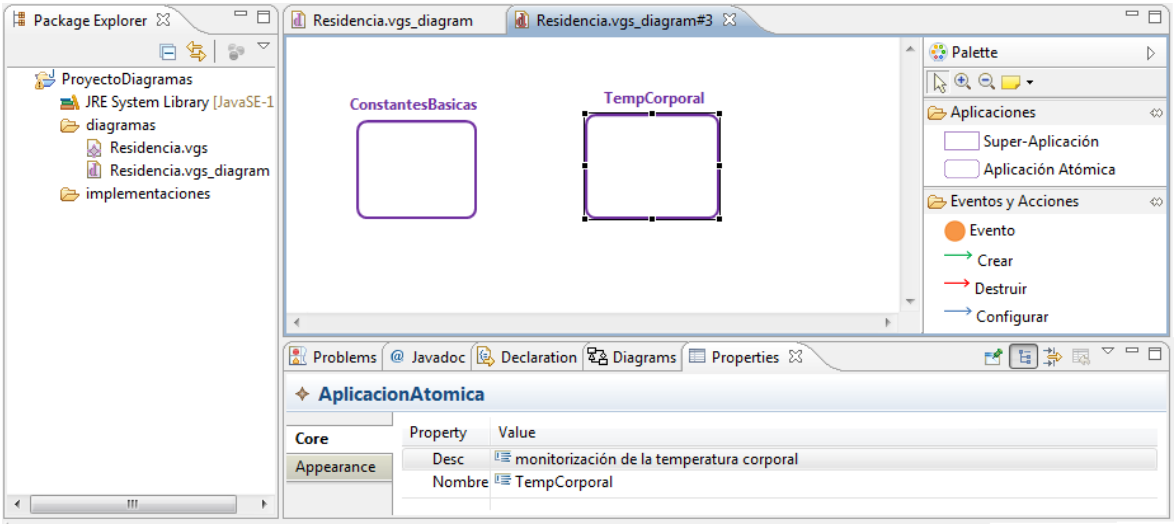


Figura 5-8: Vista Gráfica del Escenario *Residente_1*

En cuanto al escenario *Residente_2*, la supervisión de la tensión arterial se ha representado por medio de una súper-aplicación llamada *TensiónArterial* (periodo y tiempo límite de 6 horas), formada por dos aplicaciones atómicas: una para la comprobación del estado relajado (*ComprobarRelajación*, con periodo de 30 segundos y tiempo límite de 5 minutos), y otra para realizar la medida de la tensión (*MedirTensión* con tiempo límite de 5 minutos).

En el escenario *Residente_3* se ha creado una aplicación atómica (*ControlPulso*) para el control de la frecuencia cardiaca. Se trata de una aplicación que se ejecuta con periodos diferentes, en función del estado de salud del paciente. Por lo tanto, presenta un Grupo de QoS para la definición de sus requisitos temporales con dos niveles de QoS (no es QoS flexible):

- Nivel normal, con periodo y tiempo límite de 10 minutos. Es el nivel inicial.
- Nivel alarma, con periodo y tiempo límite de 1 minuto.

Por último el escenario *ZonaComún* presenta una aplicación atómica para la detección de incendios, llamada *DetecciónFuego*, con periodo y tiempo límite de 1 minuto.

3) Crear eventos, acciones asociadas y definir la propagación de eventos

En la Vista Gráfica de Escenario se crean los eventos de las aplicaciones atómicas y las acciones asociadas a dicho evento cuyo objetivo sea otra aplicación del mismo escenario.

En el caso del escenario *Residente_2*, cuando varias medidas de pulso consecutivas se encuentran alrededor de la frecuencia cardiaca en reposo del residente, se lanza el evento *Relajado* que tiene dos acciones asociadas (ver Figura 5-9): una que finaliza su propia ejecución (en color rojo) y otra para activar la aplicación *MedirTensión* (en color verde).

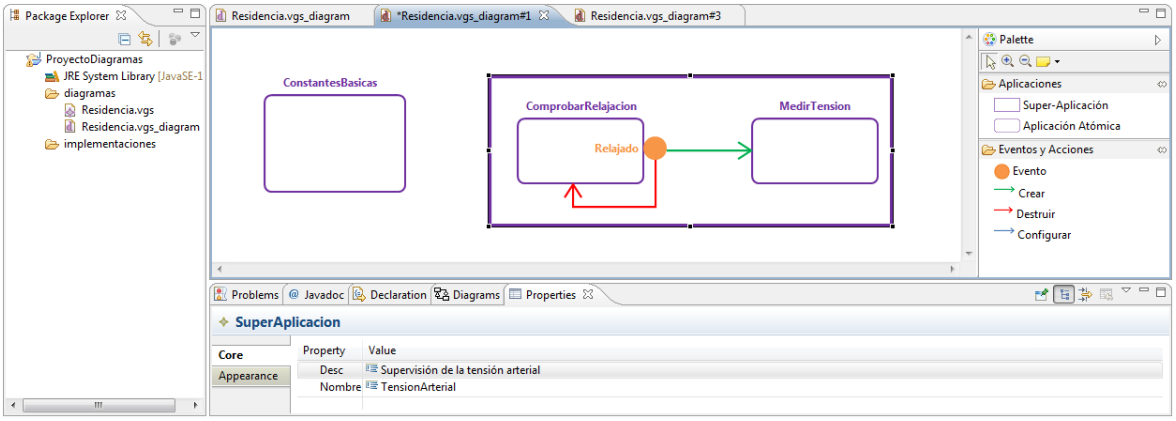


Figura 5-9: Vista Gráfica del Escenario *Residente_2*

Con respecto al escenario *Residente_3*, cuando se detecta que la frecuencia cardiaca del residente es elevada, se lanza el evento *FC_Alta* (ver Figura 5-10), cuya acción asociada, de tipo configurar, tiene como objetivo reducir el periodo y tiempo límite de la propia aplicación *ControlPulso*, es decir pasar del nivel normal al nivel alarma. Por otro lado, únicamente el personal médico puede autorizar volver al nivel normal, reiniciando la ejecución de la aplicación a través del módulo Gestor de Ejecución.

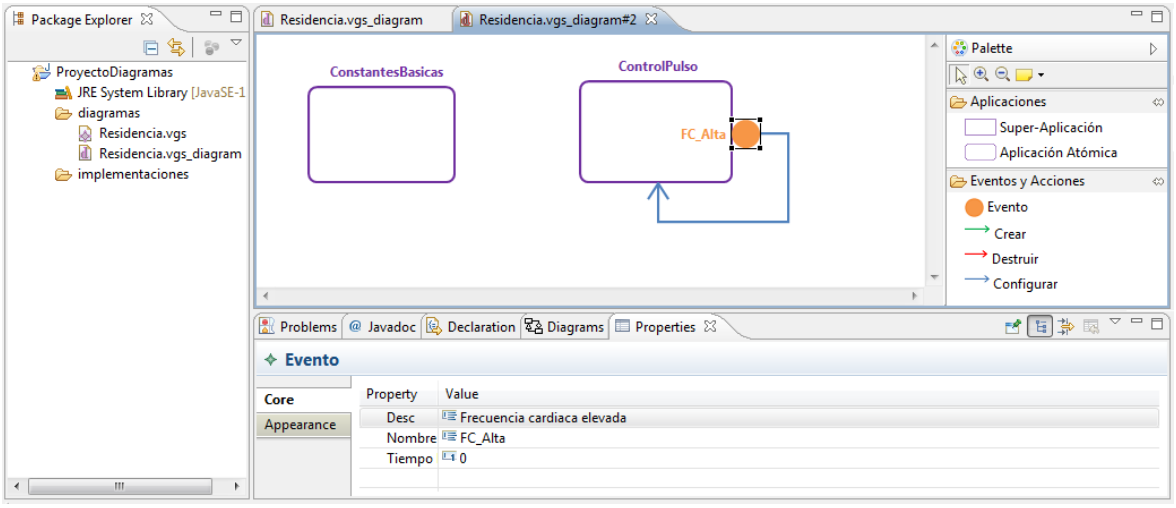


Figura 5-10: Vista Gráfica del Escenario *Residente_3*

En el escenario *ZonaComún* también se ha definido un evento, llamado *Fuego*, que representa la detección de un incendio. Sin embargo, la propagación de eventos se define en la Vista Gráfica del Sistema. Así, en la Figura 5-11 se muestra cómo el evento *Fuego* se propaga a los escenarios de los residentes con el objetivo de activar la aplicación *ConstantesBásicas* correspondiente, para lo cual se ha definido una acción de tipo crear asociada al evento recibido por cada escenario.

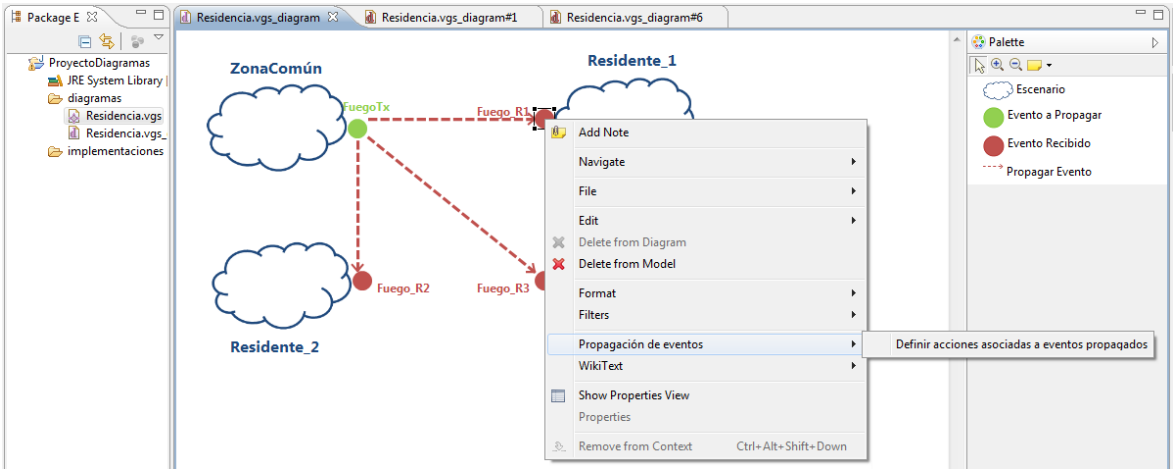


Figura 5-11: Vista Gráfica del Sistema *Residencia*: propagación de eventos

4) Especificar aplicaciones atómicas

Para la definición de todas las aplicaciones atómicas identificadas, el experto de dominio hace uso de su correspondiente Vista Gráfica de Aplicación Atómica. A modo de ejemplo, la Figura 5-12 muestra la especificación correspondiente a la aplicación

ComprobarRelajación. En ella se observan los diferentes componentes que la conforman junto con sus correspondientes puertos de entrada y/o salida, así como los conectores de datos que determinan el flujo de datos intercambiado entre ellos. Estos puertos han sido creados automáticamente por el Editor Gráfico ya que a cada componente se le ha asignado una de las unidades de servicio almacenadas en el Repositorio de Modelos.

Además, los componentes *ComprobarPulso* y *ComprobarRelajado* presentan un diagrama de actividades asociado a sus puertos de salida, para la definición de su lógica de envío de datos. Por último, el componente *ComprobarRelajado* también es el encargado del lanzamiento del evento *Relajado* según la lógica de lanzamiento asociada a su puerto de eventos.

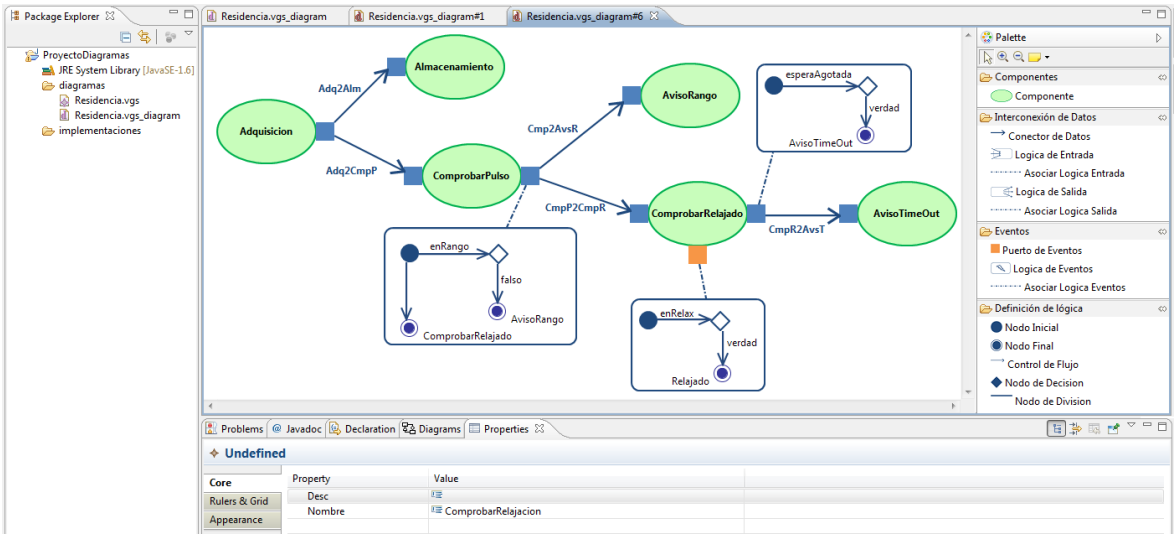


Figura 5-12: Vista Gráfica de Aplicación Atómica: *ComprobarRelajación*

5) Restricciones a nodo

Todos los componentes con unidades de servicio que controlan sensores o pantallas LCD, se han restringido al nodo que representa la Raspberry Pi correspondiente.

6) Demanda de recursos

Las implementaciones de unidad de servicio almacenadas en el Repositorio de Modelos se han caracterizado con su correspondiente demanda de recursos. Al

seleccionar la implementación deseada, en este caso la única disponible, dicha información se añade automáticamente al modelo.

7) Nivel de fiabilidad

En el caso de la plataforma de gestión MAS-RECON no es necesario indicar el nivel de fiabilidad de los componentes, ya que en tiempo de ejecución los agentes pueden viajar de un nodo a otro. Y en caso de fallo de nodo, MAS-RECON dispone de mecanismos para arrancar una nueva instancia en otro nodo, asegurando la persistencia del estado del componente, si lo precisara.

5.5 Generación Automática de Código

Para la generación de código se ha hecho uso de la implementación del módulo Generador de Código para MAS-RECON, descrita en el Capítulo 4, donde también se indica que este proceso se divide en dos: el registro en la plataforma y la obtención del código de las implementaciones de componentes.

5.5.1 Registro

Con el objetivo de ilustrar un caso práctico, la Figura 5-13 muestra la secuencia de comandos a ejecutar para registrar, en la plataforma MAS-RECON, el sistema *Residencia* del demostrador, con un solo escenario, para el *Paciente_2*, junto con sus aplicaciones atómicas (*ConstantesBásicas*, *ComprobarRelajación* y *MedirTensión*), evento (*Relajado*) y acciones asociadas. Por simplicidad, únicamente se muestra el registro de uno de sus componentes y de su correspondiente implementación. En color verde se ha resaltado la información que se extrae del modelo M_Aplicaciones.

```
reg system ID=systemResidencia name=Residencia

reg scenary ID=scenarResidente_2 name=Residente_2 parent=systemResidencia

reg application ID=applicConstantesBasicas name=ConstantesBasicas
parent=scenarResidente_2 deadline=600000

reg application ID=applicComprobarRelajacion name=ComprobarRelajacion
parent=scenarResidente_2 activation=periodic period=21600000 deadline=300000

reg application ID=applicMedirTension name=MedirTension
parent=scenarResidente_2 deadline=300000

reg event ID=eventRelajado name=Relajado parent=applicComprobarRelajacion

reg action ID=actionCrear1 name=Crear1 parent=eventRelajado order=1
action=\"start applicMedirTension\"

reg action ID=actionDestruir2 name=Destruir2 parent=eventRelajado order=2
action=\"localcmd applicComprobarRelajacion cmd=setstate stop\"

reg component ID=componAdquisicion name=Adquisicion
parent=applicComprobarRelajacion activation=periodic period=30000
deadline=30000 nodeRestriction=node201

reg cmpImplementation ID=cmpimpAdquisicionImpl name=cAdquisicionImpl
parent=componAdquisicion
class=es.ehu.generatedAgents.scnResidente_2.appComprobarRelajacion.compAdquis
icion.Adquisicion

reg qosdef ID=qosdefTemp parent=applicComprobarRelajacion
componentGroup=componAdquisicion

reg qoslev ID=qoslev1 parent=qosdefTemp

reg qosval parent=cmpimpAdquisicionImpl qoslev=qoslev1 setqoslevel=1
memory=702
```

Figura 5-13: Comandos de registro en plataforma MAS-RECON

5.5.2 Generación del código de las implementaciones de componente

En el Capítulo 4 se ha identificado un conjunto de reglas de transformación y sus correspondientes plantillas Acceleo para la plataforma de gestión MAS-RECON. Partiendo de estas plantillas y del modelo M_Aplicaciones obtenido (véase Figura 5-6) se genera el código de las implementaciones de todos los componentes definidos. En la Figura 5-14 se presenta el resultado de la generación automática de código para el estado FSM *Ejecución* correspondiente a una implementación del componente

ComprobarRelajado, que pertenece a la aplicación *ComprobarRelajación* del *Residente_2* (ver Figura 5-12). Dicho componente encapsula la unidad de servicio caracterizada en la Figura 5-3 que determina si el paciente se ha relajado, y cuya implementación se corresponde con la Figura 5-4.

```
package es.ehu.generatedAgents.scnResidente_2.appComprobarRelajacion.compComprobarRelajado.implComprobarRelajadoImpl;

import java.io.Serializable;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import jade.core.behaviours.*;
import jade.lang.acl.*;

import java.util.Date;

public class RunningBehaviour extends SimpleBehaviour {
  private static final long serialVersionUID = 5211311085804151394L;
  private MessageTemplate template;
  private ComprobarRelajado myAgent;
  private Object[] receivedMsgs;

  public RunningBehaviour(ComprobarRelajado a) {

  public RunningBehaviour(Skeleton skeleton) {

  public void onStart(){

  public void action() {
    Integer iPulso = null;
    Date dInstante = null;                                   REGLA PUERTO DE ENTRADA

    MsgCmpP2CmpR receivedData_1 = (MsgCmpP2CmpR)receivedMsgs[0];
    iPulso = receivedData_1.getiPulso();
    dInstante = receivedData_1.getdInstante();               REGLA CONECTOR DE ENTRADA

    PulseRelaxedData salida = myAgent.serviceUnit.checkPulseRelaxed(iPulso, dInstante);   REGLA UNIDAD DE SERVICIO

    if (salida.getbTimeOut() == true) {                                                   REGLA PUERTO DE SALIDA
      Serializable msg_Tx_1 = new MsgCmpRAvsT (salida.getMedia(), salida.getdTimeStamp());
      myAgent.sendMessage(msg_Tx_1, new String[] {"AvisoTimeOut" });   REGLA CONECTOR DE SALIDA
    }

    myAgent.sendState((Serializable)myAgent.serviceUnit.getComprobarRelajadoEstado());   REGLA ESTADO DE EJECUCIÓN

  }
}
```

Figura 5-14: Fragmento del código generado para la plataforma de gestión MAS-RECON

Se trata de un componente activado bajo demanda, por lo que en primer lugar se lleva a cabo la recepción de datos (regla *Puerto de Entrada*). Por cada conector de datos que llega al puerto de entrada (en este caso uno), se recibe un mensaje de datos con los parámetros establecidos en sus conexiones de datos (regla *Conector de Entrada*). Se extraen dichos parámetros de entrada para usarlos en la invocación a la unidad de servicio (regla *Unidad de Servicio*) que proporciona sus parámetros de salida. A continuación se aplica la regla *de Puerto de Salida* para interpretar el diagrama de actividades con la lógica de datos especificada. Por cada conector que sale desde dicho puerto se debe construir un mensaje de datos, en este caso uno (regla *Conector de Salida*). La regla *Puerto de Eventos* implica la interpretación del diagrama de

actividades y lanzamiento de los eventos, mientras que la regla *Acción* no es necesaria ya que MAS-RECON gestiona las acciones asociadas gracias a la información registrada en su repositorio. Por último, es un componente con estado por lo que se aplica la regla *Estado de Ejecución*.

5.6 Gestión en Ejecución

Finalmente, se arrancan las aplicaciones del demostrador. Para ello, en primer lugar es necesario compilar las implementaciones de componente generadas junto con sus correspondientes implementaciones de unidad de servicio. A continuación, se lleva a cabo el despliegue de los agentes obtenidos, que en el caso concreto de MAS-RECON, consiste en proporcionar a la plataforma de gestión este código de agente compilado. Así, cuando se solicita el arranque de una aplicación, es la propia plataforma de gestión la que despliega instancias de agentes en los nodos más adecuados en un instante concreto, gracias a la naturaleza móvil de los agentes.

5.6.1 Arranque Aplicaciones

Desde el Editor Gráfico se solicita el arranque de los escenarios que componen la residencia. A modo ilustrativo, este apartado presenta el comportamiento en ejecución de las aplicaciones del escenario *Residente_2* (ver Figura 5-9), suponiendo que son las únicas aplicaciones en ejecución. Para acotar el despliegue de instancias realizado por MAS-RECON, en el registro se han establecido las restricciones a nodo de componentes mostradas en la Tabla 5-2.

Tabla 5-2: Restricciones a nodo de los componentes del escenario *Residente_2*

APLICACIÓN	COMPONENTE	NODO
ConstantesBásicas	Adquisición	Nodo201 (Raspberry Pi)
	Almacenamiento	Nodo101 (PC), Nodo102 (PC)
ComprobarRelajación	Adquisición	Nodo201 (Raspberry Pi)
	Almacenamiento	Nodo101 (PC), Nodo102 (PC)
	ComprobarPulso	Nodo101 (PC), Nodo102 (PC)

APLICACIÓN	COMPONENTE	NODO
	AvisoRango	Nodo101 (PC), Nodo102 (PC)
	ComprobarRelajado	Nodo101 (PC), Nodo102 (PC)
	AvisoTimeOut	Nodo101 (PC), Nodo102 (PC)
MedirTensión	Adquisición	Nodo201 (Raspberry Pi)
	Almacenamiento	Nodo101 (PC), Nodo102 (PC)
	ComprobarTensión	Nodo101 (PC), Nodo102 (PC)
	AvisoRango	Nodo101 (PC), Nodo102 (PC)

La Figura 5-15 muestra el consumo de memoria a lo largo del tiempo de las instancias de agentes correspondientes a las aplicaciones del escenario *Residente_2*. En el **instante I_0** se solicita el *arranque del escenario* que consiste en el arranque de la súper-aplicación *TensiónArterial*, ya que la aplicación *ConstantesBásicas* se activa bajo demanda, como resultado de un evento propagado. Por el mismo motivo, de las dos aplicaciones definidas dentro de esta súper-aplicación únicamente se arranca la aplicación atómica *ComprobarRelajación*. Como se observa en la Figura 5-15, el arranque de las instancias conlleva la reserva de recursos correspondiente lo que provoca un aumento del consumo de memoria de los nodos en los que se despliegan, hasta el **instante I_1** en el que *finaliza el arranque* de la aplicación. El **instante I_2** se corresponde con el lanzamiento del *evento Relajación* tras detectarse que el residente se ha relajado. Como se ha definido en el apartado de modelado (ver Figura 5-12) el evento conlleva dos acciones asociadas, que se han registrado de tal manera (ver Figura 5-13) que en primer lugar se arranque la aplicación *MedirTensión* y después, **instante I_3**, se detenga la ejecución de la aplicación *ComprobarRelajación*. Es por ello que a partir del *instante I_2* se produce un aumento del consumo de memoria, que disminuye a partir del *instante I_3*. Finalmente, en el **instante I_4** *finaliza la ejecución* de la aplicación *MedirTensión*. Con objeto de observar las variaciones de memoria correctamente, se ha ralentizado tanto el proceso de arranque como de parada de las instancias de componentes.

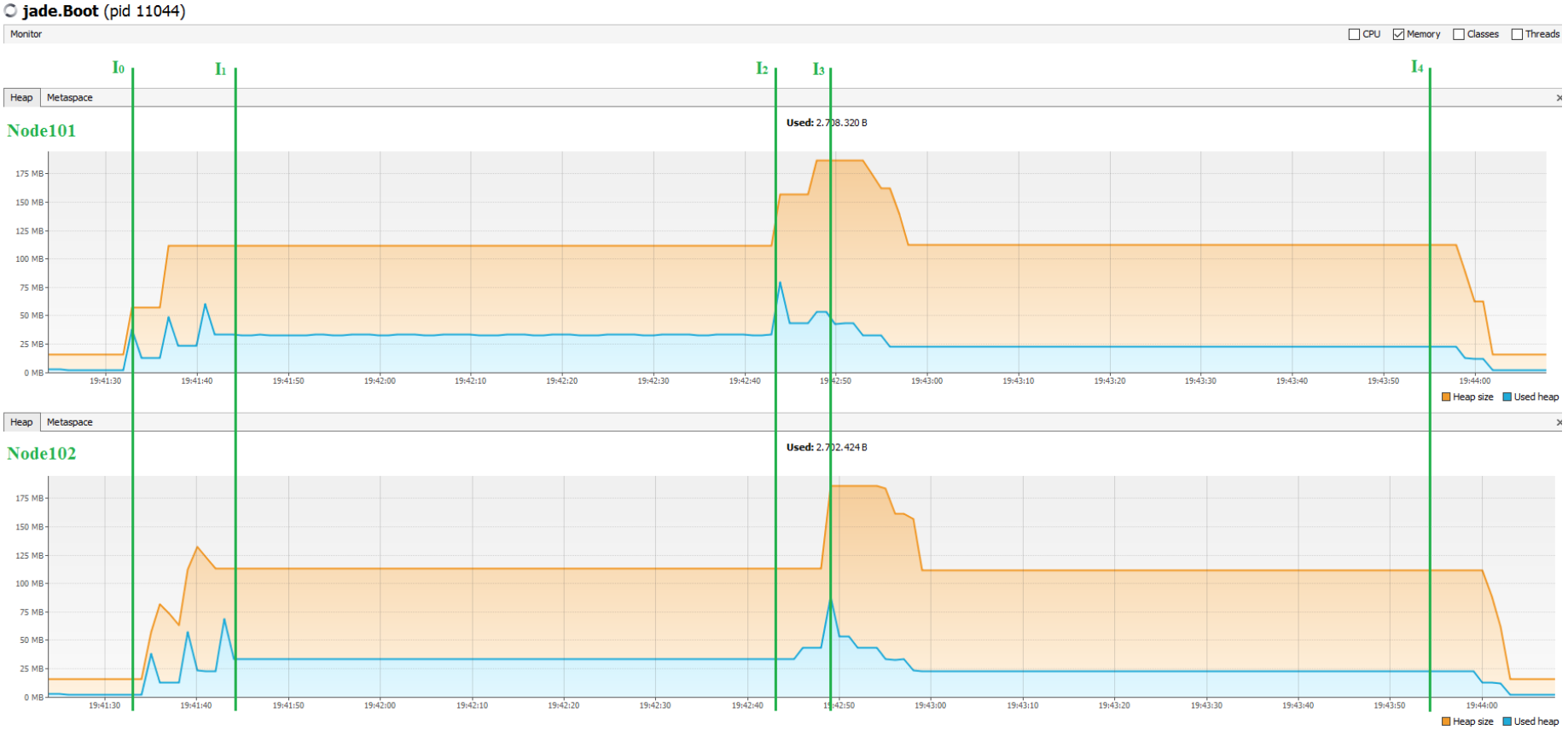


Figura 5-15: Consumo de memoria en nodos del escenario *Residente_2*

5.6.2 Monitorización en Tiempo de Ejecución

La API de la plataforma MAS-RECON dispone de la siguiente función para proporcionar información sobre el **estado del sistema** (ver Figura 5-16):

```
list runtime
```

Por lo tanto, los métodos *solicitarEstado* y *solicitarMEjecucion* de la API del Gestor de Ejecución, comprenden llamadas a esta función. Con respecto al **histórico de eventos**, la plataforma MAS-RECON proporciona la siguiente función (ver Figura 5-16):

```
list threads
```

Así, los métodos *solicitarHistórico* y *solicitarMEjecucion* de la API del Gestor de Ejecución hacen uso de esta función, seleccionando aquella información relativa al intervalo de tiempo indicado. Todas estas funciones proporcionan información en formato XML.

La Figura 5-16 muestra un detalle del resultado de la llamada a cada una de estas funciones en un instante comprendido entre I_1 e I_2 de la Figura 5-15, previo al lanzamiento del evento. Como se observa en la parte superior de la figura (*list runtime*), en dicho instante únicamente la aplicación *ComprobarRelajación* está en ejecución, por lo que en los nodos nodo101 y nodo102 sólo corren instancias de sus componentes. Con respecto a los eventos de plataforma (*list threads*) se identifican los relacionados con el arranque de la aplicación (*start*) y el paso de sus instancias por los estados FSM inicio (*boot*) y ejecución (*running*).

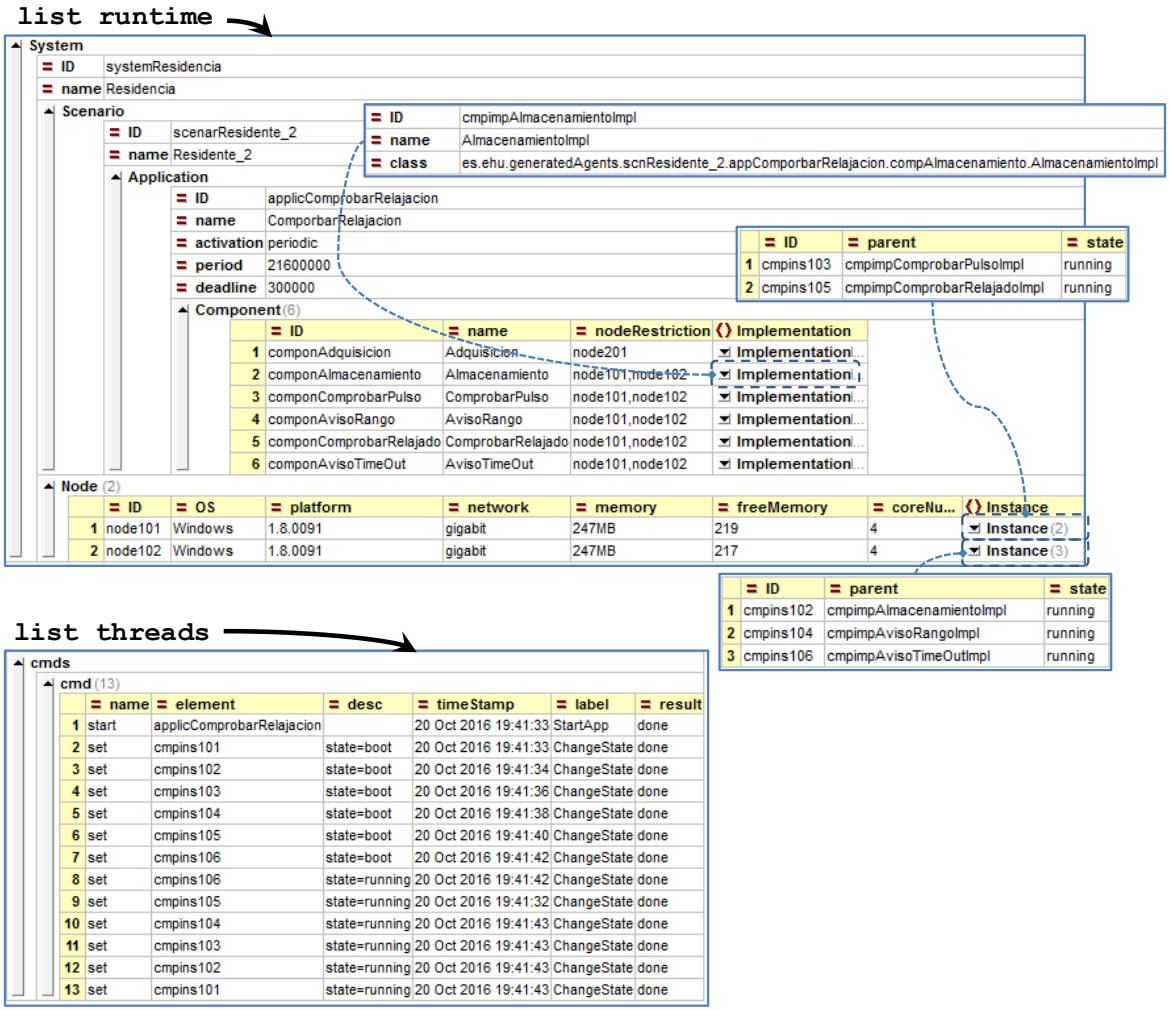


Figura 5-16: Información sobre el estado del sistema y el histórico de eventos proporcionada por MAS-RECON

El documento XML proporcionado por estas funciones de la API de MAS-RECON se debe transformar a la estructura de modelo de ejecución propuesta en el Capítulo 4, con dos objetivos: mostrar la información al operador de forma gráfica y amigable, y almacenar la información en el Repositorio de Modelos. Dicha transformación se ha llevado a cabo haciendo uso de la tecnología de hojas de estilo (*eXtensible Stylesheet Language Transformation*, XSLT) (Kelly, 2015).

Por último, para la presentación en el Editor Gráfico del modelo de ejecución obtenido, se propone un formulario con tres pestañas, tal y como se muestra en la Figura 5-17 que se corresponde con un instante de ejecución posterior al lanzamiento del evento (instante I_2 en la Figura 5-15). En la pestaña *Escenarios* se muestra el

estado de ejecución de todas las instancias de las aplicaciones activas, agrupadas por escenario. En este caso únicamente las relativas a la aplicación *MedirTensión*, arrancada tras el evento. En la pestaña *Nodos* se muestra el estado de los recursos de todos los nodos activos del sistema, mientras que en la pestaña *Histórico* se muestra la lista de eventos ordenados por instante de ocurrencia.

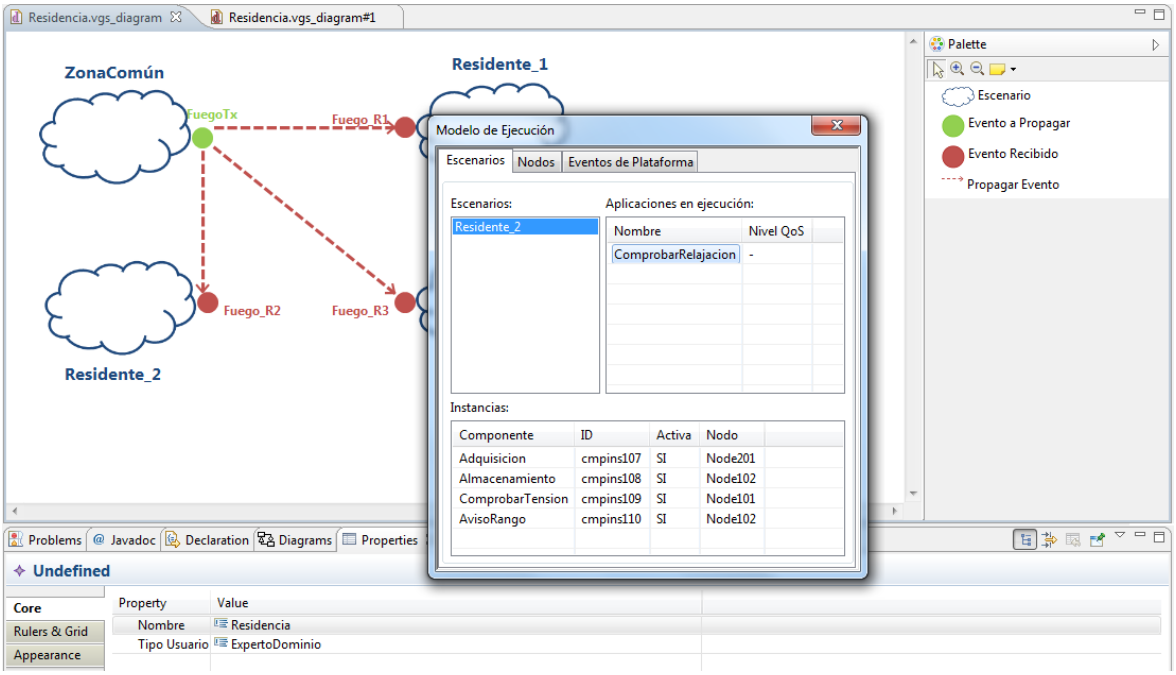


Figura 5-17: Visualización gráfica del modelo de ejecución

6 CONCLUSIONES Y LÍNEAS FUTURAS

"El futuro pertenece a quienes creen en la belleza de sus sueños."

(Amelia Earhart)

6.1 Conclusiones

La principal conclusión que se puede extraer del presente trabajo es que **es posible hacer frente a las demandas de flexibilidad** de las aplicaciones distribuidas sensibles al contexto desde las **primeras fases del diseño**. Es más, esta tarea la puede realizar, en gran medida, el experto de dominio, sin necesidad de ayuda por parte del desarrollador de software.

De hecho, la **aproximación de modelado** propuesta, **CADAMM**, permite la *separación de dominios* entre los dos expertos que participan en el diseño y desarrollo de estas aplicaciones (expertos de dominio y desarrolladores de software), *favoreciendo*, al mismo tiempo, su *colaboración*. Por un lado, CADAMM dispone de los mecanismos necesarios para que el experto de dominio realice el *diseño arquitectónico* de las aplicaciones, independiente de la implementación y plataforma de gestión. Es importante destacar que no se trata de un diseño arquitectónico al uso, ya que también contempla los requisitos de distribución, personalización, escalabilidad, QoS específica de aplicación, QoS flexible y adaptación al contexto. Por otro lado, CADAMM permite que el desarrollador se centre en el *diseño detallado e implementación* de la solución software correspondiente a dicho diseño arquitectónico, considerando también el requisito de QoS flexible.

La especificación de unidades de servicio y componentes independientes de plataforma de gestión, permite *separar la definición de la computación de la de la composición* y posibilita la *reutilización* a dos niveles: 1) nivel de aplicación, ya que una misma unidad de servicio puede ser utilizada en diferentes aplicaciones; 2) nivel de plataforma de gestión, ya que una misma unidad de servicio e incluso una misma definición aplicación puede ser empleada en diferentes plataformas de gestión. Por otro lado, CADAMM proporciona una *visión global de las interacciones entre componentes*, que en vez de orientarse al mecanismo de interacción o a los mensajes intercambiados entre ellos, se centra en los datos intercambiados. Este hecho proporciona la flexibilidad suficiente para definir *lógicas de aplicación personalizadas* a la resolución de una problemática concreta. De forma similar, esta definición de los

componentes permite también determinar *lógicas de lanzamiento de eventos* que definen la adaptación al contexto en forma de acciones a ejecutar sobre aplicaciones, solicitadas por la propia aplicación.

Además, se trata de una aproximación de modelado *genérica*, válida para cualquier aplicación distribuida sensible al contexto de cualquier dominio. Sin embargo, al mismo tiempo también puede ser *personalizada* a las particularidades concretas de un determinado campo de aplicación, e incluso *incorporar* nuevos requisitos de otros campos no contemplados.

Como consecuencia, y junto con una *plataforma de gestión* de la ejecución de las aplicaciones, permite **introducir flexibilidad en las aplicaciones**, aportándoles fundamentalmente:

- *adaptación al contexto*, identificando situaciones relevantes y reaccionando frente a ellas de forma automática, preestablecida y personalizada a cada situación.
- *ejecución distribuida*, siendo posible seleccionar los nodos óptimos que alojen sus instancias en cada momento.
- *mayor disponibilidad*, que además es *transparente a la aplicación*. No sólo debido a una gestión flexible de los recursos disponibles (adecuando la demanda de recursos a su disponibilidad, siempre que sea posible), sino también debido a la posibilidad de asegurar la consistencia de su estado tras un proceso de recuperación.
- *escalabilidad*, tanto de aplicaciones como de recursos.

Con el objetivo de dar **soporte al desarrollo** de las aplicaciones se ha propuesto un entorno integrado, **CADAMToolSuite**, que se basa en la aproximación de modelado CADAMM. Se puede concluir que CADAMToolSuite *cubre el ciclo de desarrollo* de las aplicaciones: desde su especificación a su ejecución, pasando por la generación de código. Para ello, el IDE dispone de un editor gráfico que guía al experto de dominio y al desarrollador de software en la especificación y diseño de las aplicaciones,

respectivamente. Es más, dicho editor implementa las reglas de composición de CADAMM para *asegurar la construcción de modelos correctos* mediante diversos mecanismos: proposición únicamente de opciones correctas, evitar la realización de acciones que de antemano se saben erróneas, completado automático de determinadas tareas, etc. CADAMToolSuite emplea transformaciones M2T en la generación automática de código de las aplicaciones, a partir de su modelo, y para una plataforma de gestión concreta. Además de la automatización de tareas repetitivas, el generador asegura la obtención de un *código correcto*. El IDE también es responsable de la *interacción con la plataforma de gestión*, bien para el registro bien para la monitorización de la ejecución de las aplicaciones, abstrayendo a los usuarios de las particularidades de la tecnología subyacente.

Nótese que CADAMToolSuite presenta un *diseño modular* en el que sus módulos interactúan a través del modelo de aplicaciones especificado. Por un lado, esto permite que cada módulo sea independiente de la implementación concreta del resto de módulos. Por otro lado, permite que la herramienta sea *extensible* (añadir nuevos módulos no afecta a los ya desarrollados) y *personalizable* (cambiar de plataforma de gestión o de campo de aplicación únicamente afecta a la implementación de determinados módulos).

Las aportaciones que se han descrito en esta memoria han dado lugar a un conjunto de resultados que han sido publicados en revistas con índice de impacto y presentados en conferencias internacionales de reconocido prestigio en el campo de la investigación. El trabajo realizado en relación a la *aproximación de modelado CADAMM* ha dado lugar a las siguientes publicaciones:

1. Armentia, A., Gangoiti, U., Priego, R., Estévez, E., y Marcos, M. (2015). Flexibility support for homecare applications based on models and multi-agent technology. Sensors. 15 (12), pp. 31939–31964. JCR_2015: 2.033 (Instruments & Instrumentation: T1/Q1).

2. Armentia, A., Agirre, A., Estévez, E., Pérez, J., y Marcos, M. (2014). Model Driven Design Support for Mixed-Criticality Distributed Systems. In: 19th World

Congress of the International Federation of Automatic Control (IFAC). Cape Town, South Africa: Elsevier, pp. 4441–4446.

3. Calvo, I., Portillo, E., García de Albéniz, O., Armentia, A., Marcos, M., Estévez, E., Marau, R., Almeida, L., y Pedreiras, P. (2012). Towards an Infrastructure Model for Composing and Reconfiguring Cyber-Physical Systems. In: 6th International Conference Ubiquitous Computing and Ambient Intelligence. Vitoria, Spain: Springer Berlin Heidelberg, pp. 282–289.

4. Armentia, A., Sarachaga, I., García de Albéniz, O., Estévez, E., Agirre, A., and Marcos, M. (2011). Achieving Reconfigurable Service Oriented Applications Using Model Driven Engineering. In: 16th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA). Toulouse, France: IEEE, pp. 1–4.

Con respecto al *entorno de soporte CADAMToolSuite* y la *integración del modelado con las plataformas de gestión*, se pueden destacar las siguientes publicaciones:

1. Agirre, A., Parra, J., Armentia, A., Estévez, E., y Marcos, M. (2016). QoS Aware Middleware Support for Dynamically Reconfigurable Component Based IoT Applications. International Journal of Distributed Sensor Networks. 2016 (Article ID 2702789), pp. 1–17. JCR_2015: 0.906 (Telecommunications: T2/Q3).

2. García-Valls, M., Herrasti, N., Jouvray, C. y Armentia, A. (2016). Flexible and timely on-line integration of medical services using iLand middleware. In: Medical Cyber Physical Systems Workshop 2016. Vienna, Austria.

3. Agirre, A., Parra, J., Armentia, A., Ghoneim, A., Estévez, E., y Marcos, M. (2015). QoS management for dependable sensory environments. Multimedia Tools and Applications. pp. 1-23. DOI: 10.1007/s11042-015-2781-4.

4. Armentia, A., Gangoiti, U., Priego, R., Estévez, E., y Marcos, M. (2015). A Multi-Agent Based Approach to Support Adaptability in Home Care Applications. In:

2nd IFAC Conference on Embedded Systems, Computer Intelligence and Telematics (CESCIT). Maribor, Slovenia, pp. 1–6.

Por último, durante el desarrollo del presente trabajo de investigación se han realizado colaboraciones con otros autores en la aplicación de técnicas de modelado a sistemas reconfigurables en *otros campos de aplicación*, de las cuales han resultado las siguientes publicaciones:

1. Priego, R., Armentia A., Estévez E., y Marcos, M. (2016). Modeling techniques as applied to generating tool-independent automation projects. Automatisierungstechnik. 64 (4), pp. 325–340. JCR_2015: 0.212 (AUTOMATION & CONTROL SYSTEMS: T3/Q4).

2. Priego, R., Armentia, A., Estevez, E., y Marcos, M. (2015). On applying MDE for generating reconfigurable automation systems. In: 13th IEEE International Conference on Industrial Informatics (INDIN). Cambridge, UK: IEEE, pp. 1233–1238.

3. Rafael, P., Armentia, A., Orive, D., Estévez, E., y Marcos, M. (2014). A Model-Based Approach for Achieving Available Automation Systems. In: 19th World Congress of the International Federation of Automatic Control (IFAC). Cape Town, South Africa, pp. 3438–3443.

4. Priego, R., Armentia, A., Orive, D., y Marcos, M. (2013). Supervision-based reconfiguration of industrial control systems. In: 18th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA). Cagliari, Italy: IEEE, pp. 1–4.

6.2 Líneas futuras

El trabajo desarrollado ha permitido detectar una serie de líneas de interés para futuros trabajos de investigación, centradas en aplicar los resultados obtenidos en casos reales o en otros campos de aplicación con demandas de flexibilidad. De estas líneas cabe destacar las siguientes:

- **continuar** la investigación de este tipo de aplicaciones en el campo de la **asistencia sanitaria remota o asistencia domiciliaria** mediante la *colaboración con instituciones médicas*. De esta manera se podrá ir un paso más allá de las pruebas de laboratorio, identificando nuevos requisitos no cubiertos y aplicando la aproximación de modelado CADAMM (haciendo uso del entorno CADAMToolSuite) en un caso médico real. La conexión de algunos miembros del grupo de investigación GCIS con el Aula de Ingeniería Biomédica Cruces avalada por el Hospital Universitario Cruces y el Instituto de Investigación Sanitaria Biocruces, entre otras instituciones, hacen prever una colaboración estrecha en este ámbito.

- **aplicar** los resultados obtenidos en el ámbito de la **automatización industrial**. Se trata de un campo que actualmente está inmerso en la cuarta revolución industrial caracterizada por la integración del mundo digital en los procesos productivos. El concepto industria 4.0 nace como una forma de afrontar los retos de esta cuarta revolución industrial. Ahora bien, el concepto es mucho más que internet de las cosas (IoT) o la factoría inteligente (ambos incluidos en *industria 4.0*), ya que enfatiza la idea de la consistencia en el intercambio de información entre las diferentes fases del ciclo de vida de los procesos productivos. En este contexto, parte del grupo GCIS participa en un proyecto de investigación financiado por el Ministerio de Economía y Competitividad cuyo principal objetivo consiste en proponer soluciones integradas y basadas en estándares para conseguir la integración real del proceso productivo y las áreas de negocio. Se pretenden proporcionar arquitecturas de comunicación vertical (basada en tecnología OPC UA) y horizontal (basada en agentes y capacidad de decisión), incorporando ontologías y estándares del sector que permitan desde la captura de datos del proceso productivo, procesado y filtrado de datos generando información y servicios en la nube de aplicaciones de diagnóstico del sistema de producción, hasta la toma de decisiones durante la operación. Este campo de aplicación constituye un reto en el que aplicar los resultados de este trabajo y profundizar en las particularidades propias de este campo.

- **aplicar** los resultados obtenidos en el ámbito de la **robótica colaborativa**. La robótica industrial ha aportado fiabilidad, reducción de costes y precisión a la producción en multitud de industrias. La robótica avanzada será una pieza clave en el desarrollo de máquinas autónomas. En concreto, la robótica colaborativa entra en esta categoría. Estos robots tienen la capacidad de trabajar con humanos, incluso en colaboración con ellos, debido a que tienen conciencia de su entorno. La aproximación de modelado propuesta en este trabajo puede proporcionar los mecanismos necesarios para *definir la colaboración entre dichos robots*. Cabe destacar que, por un lado, una componente del grupo GCIS (y co-directora del presente trabajo de investigación) pertenece a otro grupo de investigación con reconocido prestigio en el campo de la robótica (Grupo de Robótica, Automática y Visión por Computador (GRAV) de la Universidad de Jaén). Por otro lado, la autora del presente trabajo ha realizado una estancia de tres meses de duración con dicho grupo de investigación. Ambas situaciones han permitido establecer una colaboración entre ambos grupos en dicho ámbito.

REFERENCIAS

"Alone we can do so little; together we can do so much."

(Hellen Keller)

(Agirre et al., 2016)

Agirre, A., Parra, J., Armentia, A., Estévez, E. & Marcos, M., 2016. QoS Aware Middleware Support for Dynamically Reconfigurable Component Based IoT Applications. *International Journal of Distributed Sensor Networks*, 2016(Article ID 2702789), pp.1–17.

(Agirre et al., 2015)

Agirre, A., Parra, J., Armentia, A., Ghoneim, A., Estévez, E. & Marcos, M., 2015. QoS management for dependable sensory environments. *Multimedia Tools and Applications*, pp.1–23. DOI: 10.1007/s11042-015-2781-4

(Aiello et al., 2010)

Aiello, M., Bulanov, P. & Groefsema, H., 2010. Requirements and tools for variability management. In *34th Annual IEEE Computer Software and Applications Conference Workshops*. Seoul, South Korea: IEEE Computer Society, pp. 245–250.

(Almeida et al., 2004)

Almeida, J.P.A., van Sinderen, M., Ferreira Pires, L. & Wegdam, M., 2004. Platform-independent dynamic reconfiguration of distributed applications. In *10th IEEE International Workshop on Future Trends of Distributed Computing Systems*. Suzhou, China: IEEE, pp. 286–291.

(ArgoUML, 2001)

ArgoUML, 2001. ArgoUML. Available at: http://argouml.tigris.org/ [Accessed October 7, 2016].

(Armentia et al., 2011)

Armentia, A., Sarachaga, I., García de Albéniz, O., Estévez, E., Agirre, A. & Marcos, M., 2011. Achieving Reconfigurable Service Oriented Applications Using Model Driven Engineering. In *16th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA)*. Toulouse, France: IEEE, pp. 1–4.

(Augusto et al., 2008)

Augusto, J.C., Liu, J., McCullagh, P., Wang, H. & Yang, J.-B., 2008. Management of Uncertainty and Spatio-Temporal Aspects for Monitoring and Diagnosis in a Smart Home. *International Journal of Computational Intelligence Systems*, 1(4), pp.361–378.

(Augusto & Nugent, 2004)

Augusto, J.C. & Nugent, C.D., 2004. A New Architecture for Smart Homes Based on ADB and Temporal Reasoning. In *International Conference on Smart Home and Health Telematics (ICOST'2004)*. Singapore: Press, pp. 106–113.

(Bachmann et al., 2000)

Bachmann, F., Bass, L., Buhman, C., Comella-Dorda, S., Long, F., Robert, J., Seacord, R.C. & Wallnau, K.C., 2000. *Volume II: Technical Concepts of Component-Based Software Engineering, 2nd Edition*, Pittsburgh.

(Bajo et al., 2010)

Bajo, J., Fraile, J.A., Pérez-Lancho, B. & Corchado, J.M., 2010. The THOMAS architecture in Home Care scenarios: A case study. *Expert Systems with Applications*, 37(5), pp.3986–3999.

(Bajpai & Gorthi, 2012)

Bajpai, V. & Gorthi, R.P., 2012. On Non-Functional Requirements: A Survey. In *IEEE Students' Conference on Electrical, Electronics and Computer Science*. Bhopal, India: IEEE, pp. 1–4.

(Balasubramanian et al., 2006)

Balasubramanian, K., Gokhale, A., Karsai, G., Sztipanovits, J. & Neema, S., 2006. Developing applications using model-driven design environments. *Computer*, 39(2), pp.33–40.

(Baldauf et al., 2007)

Baldauf, M., Dustdar, S. & Rosenberg, F., 2007. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4), pp.263–277.

(Ballagny et al., 2009)

Ballagny, C., Hameurlain, N. & Barbier, F., 2009. MOCAS: A state-based component model for self-adaptation. In *3rd IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. San Francisco, USA: IEEE, pp. 206–215.

(Barkaoui et al., 2010)

Barkaoui, K., Eslamichalandar, M. & Kaabachi, M., 2010. A structural verification of web services composition compatibility. In *6th International Workshop on Enterprise & Organizational Modeling and Simulation*. Aachen, Germany: CEUR-WS.org, pp. 30–41.

(Becker, 2008)

Becker, M., 2008. Software Architecture Trends and Promising Technology for Ambient Assisted Living Systems. In *Dagstuhl Seminar Proceedings-Assisted Living Systems-Models, Architectures and Engineering Approaches*. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

(Bellavista et al., 2003)

Bellavista, P., Corradi, A. & Stefanelli, C., 2003. Application-level QoS control for video-on-demand. *IEEE Internet Computing*, 7(6), pp.16–24.

(Bellifemine et al., 2008)

Bellifemine, F., Caire, G., Poggi, A. & Rimassa, G., 2008. JADE: A software framework for developing multi-agent applications. Lessons learned. *Information and Software Technology*, 50(1–2), pp.10–21.

(Bencomo et al., 2008)

Bencomo, N., Grace, P., Flores, C., Hughes, D. & Blair, G., 2008. Genie: Supporting the model driven development of reflective, component-based adaptive systems. In *30th international conference on Software engineering*. Leipzig, Germany: ACM, pp. 811–814.

(Benghazi et al., 2012)

Benghazi, K., Hurtado, M. V., Hornos, M.J., Rodríguez, M.L., Rodríguez-Domínguez, C., Pelegrina, A.B. & Rodríguez-Fórtiz, M.J., 2012. Enabling correct design and formal analysis of Ambient Assisted Living systems. *Journal of Systems and Software*, 85(3), pp.498–510.

(Blair et al., 2009)

Blair, G., Coupaye, T. & Stefani, J.-B., 2009. Component-based architecture: The Fractal initiative. *Annals of Telecommunications*, 64(1), pp.1–4.

(Bloom, 1983)

Bloom, T., 1983. *Dynamic Module Replacement in a Distributed Programming System*. Massachussets Institute of Technology.

(Bloom & Day, 1993)

Bloom, T. & Day, M., 1993. Reconfiguration and module replacement in Argus: theory and practice. *Software Engineering Journal*, 8(2), pp.102–108.

(Booch et al., 2005)

Booch, G., Rumbaugh, J. & Jacobson, I., 2005. *The Unified Modeling Language User Guide* 2nd ed., Addison-Wesley Professional.

(Botia et al., 2012)

Botia, J.A., Villa, A. & Palma, J., 2012. Ambient Assisted Living system for in-home monitoring of healthy independent elders. *Expert Systems with Applications*, 39(9), pp.8136–8148.

(Boubeta-Puig et al., 2014)

Boubeta-Puig, J., Ortiz, G. & Medina-Bulo, I., 2014. A model-driven approach for facilitating user-friendly design of complex event patterns. *Expert Systems with Applications*, 41(2), pp.445–456.

(Bruneton et al., 2006)

Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V. & Stefani, J.-B., 2006. The FRACTAL component model and its support in Java. *Software - Practice and Experience*, 36, pp.1257–1284.

(Buccafurri et al., 2008)

Buccafurri, F., De Meo, P., Fugini, M., Furnari, R., Goy, A., Lax, G., Lops, P., Modafferi, S., Pernici, B., Redavid, D., Semeraro, G. & Ursino, D., 2008. Analysis of QoS in cooperative services for real time applications. *Data and Knowledge Engineering*, 67(3), pp.463–484.

(Bures et al., 2006)

Bures, T., Hnetynka, P. & Plasil, F., 2006. SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model. In *Fourth International Conference on Software Engineering Research, Management and Applications*. Seattle, Washington, USA: IEEE Computer Society, pp. 40–48.

(Büsching et al., 2012)

Büsching, F., Bottazzi, M. & Wolf, L., 2012. The GAL monitoring concept for distributed AAL platforms. In *IEEE 14th International Conference on e-Health Networking, Applications and Services*. Beijing, China: IEEE, pp. 466–469.

(Calvo et al., 2012)

Calvo, I., Portillo, E., García de Albéniz, O., Armentia, A., Marcos, M., Estévez, E., Marau, R., Almeida, L. & Pedreiras, P., 2012. Towards an Infrastructure Model for Composing and Reconfiguring Cyber-Physical Systems. In *6th International Conference Ubiquitous Computing and Ambient Intelligence*. Vitoria, Spain: Springer Berlin Heidelberg, pp. 282–289.

(Cano & García-Valls, 2014)

Cano, J. & García-Valls, M., 2014. Scheduling component replacement for timely execution in dynamic systems. *Software - Practice and Experience*, 44(8), pp.889–910.

(Cardellini et al., 2012)

Cardellini, V., Casalicchio, E., Grassi, V., Iannucci, S., Presti, F. Lo & Mirandola, R., 2012. MOSES: A framework for qos driven runtime adaptation of service-oriented systems. *IEEE Transactions on Software Engineering*, 38(5), pp.1138–1159.

(Cervantes & Hall, 2004)

Cervantes, H. & Hall, R.S., 2004. A Framework for Constructing Adaptive Component-Based Applications : Concepts and Experiences. In *7th International Symposium Component-Based Software Engineering (CBSE)*. Edinburgh, UK: Springer Berlin Heidelberg, pp. 130–137.

(Chalmers & Sloman, 1999)

Chalmers, D. & Sloman, M., 1999. A survey of quality of service in mobile computing environments. *IEEE Communications Surveys & Tutorials*, 2(2), pp.2–10.

(Chan et al., 2008)

Chan, M., Estève, D., Escriba, C. & Campo, E., 2008. A review of smart homes- present state and future challenges. *Computer methods and programs in biomedicine*, 91(1), pp.55–81.

(Chen, 2011)

Chen, C.-M., 2011. Web-based remote human pulse monitoring system with intelligent data analysis for home health care. *Expert Systems with Applications*, 38(3), pp.2011–2019.

(Cheng & Garlan, 2012)

Cheng, S.W. & Garlan, D., 2012. Stitch: A language for architecture-based self-adaptation. *Journal of Systems and Software*, 85(12), pp.2860–2875.

(Chung & do Prado Leite, 2009)

Chung, L. & do Prado Leite, J.C.S., 2009. On Non-Functional Requirements in Software Engineering. In A. T. Borgida, V. K. Chaudhri, P. Giorgini, & E. S. Yu, eds. *Conceptual Modeling: Foundations and Applications: Essays in Honor of John Mylopoulos*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 363–379.

(Connolly et al., 2012)

Connolly, J.F., Granger, E. & Sabourin, R., 2012. An adaptive classification system for video-based face recognition. *Information Sciences*, 192, pp.50–70.

(Cooking Hacks, 2016)

Cooking Hacks, L., 2016. Pulse and Oxygen in Blood Sensor (SPO2) for e-Health Platform. Available at: https://www.cooking-hacks.com/pulse-and-oxygen-in-blood-sensor-spo2-ehealth-medical [Accessed June 9, 2016].

(Cooking Hacks, 2013a)

Cooking Hacks, 2013. e-Health Sensor Platform V2.0 for Arduino and Raspberry Pi [Biometric / Medical Applications]. Available at: https://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical [Accessed October 13, 2015].

(Cooking Hacks, 2013b)

Cooking Hacks, 2013. Waspmote Gas Sensors Kit. Available at: https://www.cooking-hacks.com/shop/waspmote/kits/waspmote-gas-sensors-kit [Accessed October 13, 2015].

(Corchado et al., 2008)

Corchado, J.M., Bajo, J. & Abraham, A., 2008. GerAmi: Improving Healthcare Delivery in Geriatric Residences. *IEEE Intelligent Systems*, 23(2), pp.19–25.

(Crnkovic et al., 2011)

Crnkovic, I., Sentilles, S., Vulgarakis, A. & Chaudron, M.R. V., 2011. A Classification Framework for Software Component Models. *IEEE Transactions on Software Engineering*, 37(5), pp.593–615.

(Cuevas, 2016)

Cuevas, C., 2016. *Metaherramientas MDE para el diseño de entornos de desarrollo de sistemas distribuidos de tiempo real*. Universidad de Cantabria.

(Cuevas et al., 2016a)

Cuevas, C., López, P. & Drake, J.M., 2016. Automating the Construction of Models based on Domain Views. In *4th International Conference on Model-Driven Engineering and Software Development*. Rome, Italy, pp. 241–249.

(Cuevas et al., 2016b)

Cuevas, C., López, P. & Drake, J.M., 2016. MDDE : Una concepción genérica para diseño de entornos de desarrollo de software basados en MDSE. In *XXI Jornadas de Ingeniería del Software y Bases de Datos - CEDI 2016*. Salamanca, Spain: Ediciones Universidad Salamanca, pp. 241–254.

(Davis, 1993)

Davis, A.M., 1993. *Software requirements: objects, functions, and states* 2nd ed., Prentice-Hall, Inc.

(Dey, 2001)

Dey, A.K., 2001. Understanding and Using Context. *Personal and Ubiquitous Computing 5*, 5(1), pp.4–7.

(Dijkman & Dumas, 2004)

Dijkman, R. & Dumas, M., 2004. Service-Oriented Design: a Multi-Viewpoint Approach. *International Journal of Cooperative Information Systems*, 13(4), pp.337–368.

(Dowling & Cahill, 2001)

Dowling, J. & Cahill, V., 2001. Dynamic Software Evolution and The K-Component Model. In *Workshop on Software Evolution (OOPSLA 2001)*. Tampa Bay, Florida, USA.

(Dumez et al., 2013)

Dumez, C., Bakhouya, M., Gaber, J., Wack, M. & Lorenz, P., 2013. Model-driven approach supporting formal verification for web service composition protocols. *Journal of Network and Computer Applications*, 36(4), pp.1102–1115.

(Dunkel et al., 2011)

Dunkel, J., Fernández, A., Ortiz, R. & Ossowski, S., 2011. Event-driven architecture for decision support in traffic management systems. *Expert Systems with Applications*, 38(6), pp.6530–6539.

(Eclipse, 2010a)

Eclipse, 2010. Graphical Modeling Project (GMP). Available at: http://www.eclipse.org/modeling/gmp/ [Accessed October 5, 2016].

(Eclipse, 2010b)

Eclipse, 2010. Java Emitter Templates (JET). Available at: http://www.eclipse.org/modeling/m2t/?project=jet [Accessed October 5, 2016].

(Eclipse Epsilon Project, 2009)

Eclipse Epsilon Project, 2009. EuGENia. Available at: http://www.eclipse.org/epsilon/doc/eugenia/ [Accessed August 13, 2016].

(Eclipse Modeling Project, 2014)

Eclipse Modeling Project, 2014. Graphiti. Available at: http://www.eclipse.org/graphiti/.

(Egbogah & Fapojuwo, 2011)

Egbogah, E.E. & Fapojuwo, A.O., 2011. A survey of system architecture requirements for health care-based wireless sensor networks. *Sensors*, 11(5), pp.4875–4898.

(Eichelberg et al., 2010)

Eichelberg, M., Hein, A., Büsching, F. & Wolf, L., 2010. The GAL middleware platform for AAL: A Case Study. In *12th IEEE International Conference on e-Health Networking Applications and Services*. Lyon, France: IEEE, pp. 1–6.

(Erl, 2005)

Erl, T., 2005. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*, Prentice Hall PTR.

(Espinoza et al., 2006)

Espinoza, H., Dubois, H., Gérard, S., Medina, J., Petriu, D.C. & Woodside, M., 2006. Annotating UML Models with Non-functional Properties for Quantitative Analysis. In *MoDELS 2005 International Workshops Doctoral Symposium, Educators Symposium*. Montego Bay, Jamaica: Springer Berlin Heidelberg, pp. 79–90.

(Estublier & Vega, 2012)

Estublier, J. & Vega, G., 2012. Reconciling components and services: The Apam component-service platform. *Proceedings - 2012 IEEE 9th International Conference on Services Computing, SCC 2012*, pp.683–684.

(Farella et al., 2010)

Farella, E., Falavigna, M. & Ricc, B., 2010. Aware and smart environments: The Casattenta project. *Microelectronics Journal*, 41(11), pp.697–702.

(Fiadeiro & Lopes, 2013)

Fiadeiro, J.L. & Lopes, A., 2013. An interface theory for service-oriented design. *Theoretical Computer Science*, 503(9), pp.1–30.

(Fleurey & Solberg, 2009)

Fleurey, F. & Solberg, A., 2009. A Domain Specific Modeling Language supporting Specification, Simulation and Execution of Dynamic Adaptive Systems. In *12th International Conference on Model-Driven Engineering Languages and Systems*. Denver, USA: Springer Berlin Heidelberg, pp. 606–621.

(Foster et al., 2011)

Foster, H., Mukhija, A., Rosenblum, D.D. & Uchitel, S., 2011. Specification and Analysis of Dynamically-Reconfigurable Service Architectures. In M. Wirsing & M. Hölzl, eds. *Rigorous Software Engineering for Service-Oriented Systems*. Berlin, Heidelberg: Springer-Verlag, pp. 428–446.

(Foundation for Intelligent Physical Agents, 2002)

Foundation for Intelligent Physical Agents, 2002. Standard FIPA specifications. Available at: http://www.fipa.org/repository/standardspecs.html [Accessed September 9, 2016].

(Frølund & Koistinen, 1999)

Frølund, S. & Koistinen, J., 1999. Quality of service aware distributed object systems. In *5th USENIX Conference on Object-Oriented Technologies & Systems*. San Diego, USA: USENIX Association Berkeley, pp. 1–15.

(Galster et al., 2014)

Galster, M., Weyns, D., Tofan, D., Michalik, B. & Avgeriou, P., 2014. Variability in Software Systems: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 40(3), pp.282–306.

(Gangoiti et al., 2016)

Gangoiti, U., Armentia, A., Priego, R., Estévez, E. & Marcos, M., 2016. MAS-RECON. MIDDLEWARE RECONFIGURABLE BASADO EN MULTIAGENTES. In *XXXVII JORNADAS DE AUTOMÁTICA*. Madrid, Spain.

(García-Magariño & Gutiérrez, 2013)

García-Magariño, I. & Gutiérrez, C., 2013. Agent-oriented modeling and development of a system for crisis management. *Expert Systems with Applications*, 40(16), pp.6580–6592.

(García-Valls & Basanta-Val, 2013)

García-Valls, M. & Basanta-Val, P., 2013. A real-time perspective of service composition: Key concepts and some contributions. *Journal of Systems Architecture*, 59(10 PART D), pp.1414–1423.

(García-Valls et al., 2014)

García-Valls, M., Uriol-Resuela, P., Ibáñez-Vázquez, F. & Basanta-Val, P., 2014. Low complexity reconfiguration for real-time data-intensive service-oriented applications. *Future Generation Computer Systems*, 37, pp.191–200.

(García-Valls et al., 2013a)

García-Valls, M., Basanta-Val, P., Marcos, M. & Estévez, E., 2013. A bi-dimensional QoS model for SOA and real-time middleware. *Computer Systems Science and Engineering*, 28(4), pp.1–11.

(García-Valls et al., 2013b)

García-Valls, M., Rodríguez-López, I. & Fernández-Villar, L., 2013. iLAND : An Enhanced Middleware for Real-Time Reconfiguration of Service Oriented Distributed Real-Time Systems. *IEEE Transactions on Industrial Informatics*, 9(1), pp.228–236.

(García-Valls et al., 2012)

García-Valls, M., Alonso, A. & De La Puente, J.A., 2012. A dual-band priority assignment algorithm for dynamic QoS resource management. *Future Generation Computer Systems*, 28(6), pp.902–912.

(Gharzouli & Boufaida, 2009)

Gharzouli, M. & Boufaida, M., 2009. A generic P2P collaborative strategy for discovering and composing semantic web services. In *4th International Conference on Internet and Web Applications and Services*. Venice/Mestre, Italy, pp. 449–454.

(Glinz, 2007)

Glinz, M., 2007. On Non-Functional Requirements. In *15th IEEE International Requirements Engineering Conference*. Delhi, India: IEEE, pp. 21–26.

(Goupil, 2010)

Goupil, P., 2010. Oscillatory failure case detection in the A380 electrical flight control system by analytical redundancy. *Control Engineering Practice*, 18(9), pp.1110–1119.

(Gronback, 2009)

Gronback, R.C., 2009. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit* 1st ed., Addison-Wesley Professional.

(Gui et al., 2011)

Gui, N., De Florio, V., Sun, H. & Blondia, C., 2011. Toward architecture-based context-aware deployment and adaptation. *Journal of Systems and Software*, 84(2), pp.185–197.

(Hallsteinsen et al., 2012)

Hallsteinsen, S., Geihs, K., Paspallis, N., Eliassen, F., Horn, G., Lorenzo, J., Mamelli, A. & Papadopoulos, G.A., 2012. A development framework and methodology for self-adapting applications in ubiquitous computing environments. *Journal of Systems and Software*, 85(12), pp.2840–2859.

(Hammer, 2009)

Hammer, M., 2009. *How To Touch a Running System - Reconfiguration of Stateful Components*. Ludwig-Maximilians-Universit¨ at M¨ unchen zur.

(Hammer & Knapp, 2010)

Hammer, M. & Knapp, A., 2010. Correct Execution of Reconfiguration for Stateful Components. *Electronic Notes in Theoretical Computer Science*, 260, pp.91–108.

(Harbour et al., 2002)

Harbour, M.G., Medina, J.L., Gutiérrez, J.J., Palencia, J.C. & Drake, J.M., 2002. MAST : An Open Environment for Modeling , Analysis , and Design of Real-Time Systems. In *1st CARTS Workshop*. Aranjuez, Spain, pp. 1–16.

(Hassine, 2015)

Hassine, J., 2015. Describing and assessing availability requirements in the early stages of system development. *Software and Systems Modeling*, 14(4), pp.1455–1479.

(Heineman & Councill, 2001)

Heineman, G.T. & Councill, W.T., 2001. *Component-based software engineering: putting the pieces together* 1st ed., Addison-Wesley Longman Publishing Co., Inc.

(Heineman & Councill, 2001)

Hervás, R., Bravo, J. & Fontecha, J., 2010. A Context Model based on Ontological Languages : a Proposal for Information Visualization. *Journal of Universal Computer Science*, 16(12), pp.1539–1555.

(Hervás et al., 2010)

Hervás, R., Fontecha, J., Ausín, D., Castanedo, F., Bravo, J. & López-de-Ipiña, D., 2013. Mobile monitoring and reasoning methods to prevent cardiovascular diseases. *Sensors*, 13(5), pp.6524–6541.

(Hofmeister, 1998)

Hofmeister, C.R., 1998. *Dynamic Reconfiguration of Distributed Applications*. University of Maryland.

(Holborn et al., 2003)

Holborn, P.., Nolan, P.. & Golt, J., 2003. An analysis of fatal unintentional dwelling fires investigated by London Fire Brigade between 1996 and 2000. *Fire Safety Journal*, 38(1), pp.1–42.

(Horling & Lesser, 2004)

Horling, B. & Lesser, V., 2004. A Survey of Multi-Agent Organizational Paradigms. *The Knowledge Engineering Review*, 19(4), pp.281–316.

(Hoyos et al., 2013)

Hoyos, J.R., García-Molina, J. & Botía, J.A., 2013. A domain-specific language for context modeling in context-aware systems. *Journal of Systems and Software*, 86(11), pp.2890–2905.

(IEEE, 1990)

IEEE, 1990. IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990.

(Intrieri et al., 2012)

Intrieri, E., Gigli, G., Mugnai, F., Fanti, R. & Casagli, N., 2012. Design and implementation of a landslide early warning system. *Engineering Geology*, 147–148, pp.124–136.

(Isern et al., 2011)

Isern, D., Sánchez, D. & Moreno, A., 2011. Organizational structures supported by agent-oriented methodologies. *Journal of Systems and Software*, 84(2), pp.169–184.

(ISO/IEC, 2014)

ISO/IEC, 2014. ISO/IEC 25000 SQuaRE series. Available at: https://www.iso.org/obp/ui/#iso:std:iso-iec:25000:ed-2:v1:en [Accessed October 1, 2016].

(Jain et al., 2006)

Jain, A.K., Ross, A. & Pankanti, S., 2006. Biometrics: A tool for information security. *IEEE Transactions on Information Forensics and Security*, 1(2), pp.125–143.

(Jobbágy et al., 2006)

Jobbágy, Á., Csordás, P. & Mersich, A., 2006. Blood Pressure Measurement at Home. In R. Magjarevic & J. H. Nagel, eds. *2006 World Congress on Medical Physics and Biomedical Engineering*. Seoul, Korea (South): Springer Berlin Heidelberg, pp. 3453–3456.

(Jureta et al., 2006)

Jureta, I.J., Faulkner, S. & Schobbens, P.-Y., 2006. A More Expressive Softgoal Conceptualization for Quality Requirements Analysis. In *25th International Conference on Conceptual Modeling*. Tucson, USA: Springer Berlin Heidelberg, pp. 281–295.

(Kelly, 2015)

Kelly, D.J., 2015. *XSLT Jumpstarter*, Pragmatic Bookshelf.

(Khan et al., 2008)

Khan, M.U., Reichle, R. & Geihs, K., 2008. Architectural constraints in the model-driven development of self-adaptive applications. *IEEE Distributed Systems Online*, 9(7), pp.1–10.

(Kon et al., 2005)

Kon, F., Marques, J.R., Yamane, T., Campbell, R.H. & Mickunas, M.D., 2005. Design, implementation, and performance of an automatic configuration service for distributed component systems. *Software: Practice and Experience*, 35(7), pp.667–703.

(Kramer & Magee, 1990)

Kramer, J. & Magee, J., 1990. The evolving philosophers problem: dynamic change management. *IEEE Transactions on Software Engineering*, 16(11), pp.1293–1306.

(Krupitzer et al., 2015)

Krupitzer, C., Roth, F.M., VanSyckel, S., Schiele, G. & Becker, C., 2015. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17(Part B), pp.184–206.

(Landes & Studer, 1995)

Landes, D. & Studer, R., 1995. The treatment of non-functional requirements in MIKE. In *5th European Software Engineering Conference*. Sitges, Spain: Springer Berlin Heidelberg, pp. 294–306.

(Lau & Wang, 2007)

Lau, K.-K. & Wang, Z., 2007. Software Component Models. *IEEE Transactions on Software Engineering*, 33(10), pp.709–724.

(Laws et al., 2011)

Laws, S., Combellack, M., Feng, R., Mahbod, H. & Nash, S., 2011. *Tuscany SCA in Action* 1st ed., Greenwich, CT, USA: Manning Publications Co.

(Léger et al., 2010)

Léger, M., Ledoux, T. & Coupaye, T., 2010. Reliable Dynamic Reconfigurations in a Reflective Component Model. In *13th international conference on Component-Based Software Engineering*. Prague, Czech Republic: Springer Berlin Heidelberg, pp. 74–92.

(Li & Nahrstedt, 1999)

Li, B. & Nahrstedt, K., 1999. A Control-Based Middleware Framework for Quality-of-Service Adaptations. *IEEE Journal on Selected Areas in Communications*, 17(9), pp.1632–1650.

(Li, 2012)

Li, W., 2012. QoS assurance for dynamic reconfiguration of component-based software systems. *IEEE Transactions on Software Engineering*, 38(3), pp.658–676.

(Li, 2011)

Li, W., 2011. Evaluating the impacts of dynamic reconfiguration on the QoS of running systems. *Journal of Systems and Software*, 84(12), pp.2123–2138.

(Li & Guo, 2015)

Li, W. & Guo, W., 2015. QoS prediction for dynamic reconfiguration of component based software systems. *Journal of Systems and Software*, 102, pp.12–34.

(Lim et al., 2014)

Lim, M.K., Tang, S. & Chan, C.S., 2014. ISurveillance: Intelligent framework for multiple events detection in surveillance videos. *Expert Systems with Applications*, 41(10), pp.4704–4715.

(Liu et al., 2006)

Liu, J., Augusto, J.C. & Wang, H., 2006. Considerations on Uncertain Spatio-Temporal Reasoning in Smart Home Systems. *7th International FLINS Conference*, pp.817–824.

(López et al., 2013)

López, P., Barros, L. & Drake, J.M., 2013. Design of component-based real-time applications. *Journal of Systems and Software*, 86(2), pp.449–467.

(Loyall et al., 1998)

Loyall, J., Bakken, D.E., Schantz, R.E., Zinky, J.A., Karr, D.A., Vanegas, R. & Anderson, K.R., 1998. QoS Aspect Languages and Their Runtime Integration. In *4th International Workshop LCR '98*. Pittsburgh, USA: Springer Berlin Heidelberg, pp. 303–318.

(Macías-Escrivá et al., 2013)

Macías-Escrivá, F.D., Haber, R., del Toro, R. & Hernandez, V., 2013. Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40(18), pp.7267–7279.

(Mairiza et al., 2010)

Mairiza, D., Zowghi, D. & Nurmuliani, N., 2010. An Investigation into the Notion of Non-Functional Requirements. In *2010 ACM Symposium on Applied Computing*. Sierre, Switzerland: ACM New York, pp. 311–317.

(Mitchell et al., 1999)

Mitchell, S., Naguib, H., Coulouris, G. & Kindberg, T., 1999. A Qos Support Framework for Dynamically Multimedia Applications. In *2nd International Working Conference on Distributed Applications and Interoperable Systems*. Helsinki, Finland: Springer US, pp. 17–30.

(Morin et al., 2009)

Morin, B., Barais, O., Jezequel, J.-M., Fleurey, F. & Solberg, A., 2009. Models@ Run.time to Support Dynamic Adaptation. *Computer*, 42(10), pp.44–51.

(Nehmer et al., 2006)

Nehmer, J., Becker, M., Karshmer, A. & Lamm, R., 2006. Living Assistance Systems - An Ambient Intelligence Approach -. In *Proceeding of the 28th international conference on Software engineering*. Shanghai, China: ACM Press, pp. 43–50.

(Network Time Foundation, 2016)

Network Time Foundation, 2016. NTP Project. Available at: http://nwtime.org/projects/ntp/ [Accessed July 5, 2016].

(Noguero & Calvo, 2012)

Noguero, A. & Calvo, I., 2012. FTT-Modeler: A support tool for FTT-CORBA. *Information Systems and Technologies (CISTI), 2012 7th Iberian Conference on*, pp.1–6.

(Noguero et al., 2013)

Noguero, A., Calvo, I., Pérez, F. & Almeida, L., 2013. FTT-MA: a flexible time-triggered middleware architecture for time sensitive, resource-aware AmI systems. *Sensors*, 13(5), pp.6229–6253.

(Noran, 2014)

Noran, O., 2014. Collaborative disaster management: An interdisciplinary approach. *Computers in Industry*, 65(6), pp.1032–1040.

(OASIS, 2007)

OASIS, 2007. Service Component Architecture (SCA). Available at: http://www.oasis-opencsa.org/sca [Accessed July 5, 2016].

(Obeo Network, 2006)

Obeo Network, 2006. Acceleo Project. Available at: http://www.eclipse.org/acceleo/ [Accessed August 13, 2016].

(Oldevik et al., 2005)

Oldevik, J., Neple, T., Grønmo, R., Aagedal, J. & Berre, A.-J., 2005. Toward Standardised Model to Text Transformations. In *First European Conference, ECMDA-FA*. Nuremberg, Germany: Springer Berlin Heidelberg, pp. 239–253.

(OMG, 2016)

OMG, Object Management Group, 2016. Interface Definition Language™ (IDL™) 4.0. *Formal Version of IDL*, p.130. Available at: http://www.omg.org/spec/IDL/ [Accessed June 9, 2016].

(OMG, 2015)

OMG, Object Management Group, 2015. XML Metadata Interchange Version 2.5.1. Available at: http://www.omg.org/spec/XMI/2.5.1/ [Accessed October 5, 2016].

(OMG, 2012)

OMG, Object Management Group, 2012. Service Oriented Architecture Modeling Language™ (SoaML®). Available at: http://www.omg.org/spec/SoaML/ [Accessed September 6, 2016].

(OMG, 2011)

OMG, Object Management Group, 2011. UML Profile for MARTE : Modeling and Analysis of Real-Time Embedded Systems V1.1.

(OMG, 2008)

OMG, Object Management Group, 2008. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms V1.1.

(OMG, 2007)

OMG, Object Management Group, 2007. Data Distribution Service™, v1.2. Available at: http://www.omg.org/spec/DDS/1.2/ [Accessed July 5, 2016].

(OMG, 2005)

OMG, Object Management Group, 2005. UML Profile for Schedulability , Performance, and Time Specification V1.1.

(Papazoglou et al., 2007)

Papazoglou, M.P., Traverso, P., Dustdar, S. & Leymann, F., 2007. Service-Oriented Computing : State of the Art and Research Challenges. *Computer*, 40(11), pp.38–45.

(Pérez et al., 2009)

Pérez, F., Valderas, P. & Fons, J., 2009. Enabling end-users participation in an MDD-SPL approach. In *1st International Workshop on Model-driven Approaches in Software Product Line Engineering (MAPLE 2009)*. San Francisco, USA, pp. 6–13.

(Qiu et al., 2017)

Qiu, X., Ali, S., Yue, T. & Zhang, L., 2017. Reliability-redundancy-location allocation with maximum reliability and minimum cost using search techniques. *Information and Software Technology*, 82, pp.36–54.

(Rabbi et al., 2014)

Rabbi, F., Lamo, Y. & Maccaull, W., 2014. A Flexible Metamodelling Approach for Healthcare Systems. In *2nd European Workshop on Practical Aspects of Health Informatics*. Trondheim, Norway, pp. 115–128.

(Ramesh, 2014)

Ramesh, M.V., 2014. Design, development, and deployment of a wireless sensor network for detection of landslides. *Ad Hoc Networks*, 13(PART A), pp.2–18.

(Rocha et al., 2013)

Rocha, A., Martins, A., Freire, J.C., Kamel Boulos, M.N., Vicente, M.E., Feld, R., van de Ven, P., Nelson, J., Bourke, A., ÓLaighin, G., Sdogati, C., Jobes, A., Narvaiza, L. & Rodríguez-Molinero, A., 2013. Innovations in health care services: The CAALYX system. *International Journal of Medical Informatics*, 82(11), pp.e307–e320.

(Roman, 1985)

Roman, G.C., 1985. Taxonomy of Current Issues in Requirements Engineering. *Computer*, 18(4), pp.14–23.

(Sadri, 2011)

Sadri, F., 2011. Ambient intelligence: A Survey. *ACM Computing Surveys*, 43(4), pp.1–66.

(Salazar et al., 2014)

Salazar, E., Alonso, A. & Garrido, J., 2014. Mixed-criticality design of a satellite software system. In *19th World Congress of the International Federation of Automatic Control (IFAC)*. Cape Town, South Africa: IFAC-PapersOnLine, pp. 12278–12283.

(Salazar et al., 2013)

Salazar, E., Alonso, A., De Miguel, M.A. & De La Puente, J.A., 2013. A Model-Based Framework for Developing Real-Time Safety Ada Systems. In *18th Ada-Europe International Conference on Reliable Software Technologies*. Springer Berlin Heidelberg, pp. 127–142.

(Salehie & Tahvildari, 2009)

Salehie, M. & Tahvildari, L., 2009. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2), p.14:1-14:42.

(Sánchez et al., 2011)

Sánchez, P., Jiménez, M., Rosique, F., Álvarez, B. & Iborra, A., 2011. A framework for developing home automation systems: From requirements to code. *Journal of Systems and Software*, 84(6), pp.1008–1021.

(Schilit & Theimer, 1994)

Schilit, B.N. & Theimer, M.M., 1994. Disseminating active map information to mobile hosts. *IEEE Network*, 8(5), pp.22–32.

(Schmidt, 2006)

Schmidt, D.C., 2006. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2), pp.25–31.

(Seinturier et al., 2009)

Seinturier, L., Merle, P., Fournier, D., Dolet, N., Schiavoni, V. & Stefani, J.-B., 2009. Reconfigurable SCA Applications with the FraSCAti Platform. In *IEEE International Conference on Services Computing*. Bangalore, India: IEEE Computer Society, pp. 268–275.

(Selic, 2008)

Selic, B., 2008. Personal reflections on automation, programming culture, and model-based software engineering. *Automated Software Engineering*, 15(3), pp.379–391.

(Selic, 2003)

Selic, B., 2003. The pragmatics of model-driven development. *IEEE Software*, 20(5), pp.19–25.

(Sentilles et al., 2009)

Sentilles, S., Štěpán, P., Carlson, J. & Crnković, I., 2009. Integration of Extra-Functional Properties in Component Models. In *12th International Symposium, CBSE 2009*. East Stroudsburg, USA: Springer Berlin Heidelberg, pp. 173–190.

(Sheng & Benatallah, 2005)

Sheng, Q.Z. & Benatallah, B., 2005. ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services Development. In *International Conference on Mobile Business (ICMB'05)*. Sydney, Australia: IEEE, pp. 206–212.

(Siegel & Retter, 2014)

Siegel, E. & Retter, A., 2014. *eXist: A NoSQL Document Database and Application Platform*, O'Reilly Media.

(De Silva et al., 2012)

De Silva, L.C., Morikawa, C. & Petra, I.M., 2012. State of the art of smart homes. *Engineering Applications of Artificial Intelligence*, 25(7), pp.1313–1321.

(Skogan et al., 2004)

Skogan, D., Gronmo, R. & Solheim, I., 2004. Web service composition in UML. In *8th IEEE International Enterprise Distributed Object Computing Conference*. Vienna, Austria: IEEE, pp. 47–57.

(Søberg et al., 2010)

Søberg, J., Goebel, V. & Plagemann, T., 2010. CommonSens: Personalisation of complex event processing in automated homecare. In *6th International Conference on Intelligent Sensors, Sensor Networks and Information Processing*. Brisbane, Australia: IEEE, pp. 275–280.

(de Souza Neto, 2012)

de Souza Neto, P.A., 2012. *A Methodology for Building Service-Oriented Applications in the Presence of Non-Functional Properties*. Universidade Federal do Rio Grande do Norte.

(Sparx Systems, 2016)

Sparx Systems, 2016. Enterprise Architect. Available at: http://www.sparxsystems.com/products/ea/index.html [Accessed August 13, 2016].

(Stav et al., 2013)

Stav, E., Walderhaug, S., Mikalsen, M., Hanke, S. & Benc, I., 2013. Development and evaluation of SOA-based AAL services in real-life environments: A case study and lessons learned. *International journal of medical informatics*, 82(11), pp.e269–e293.

(Steinberg et al., 2008)

Steinberg, D., Budinsky, F., Paternostro, M. & Merks, E., 2008. *EMF : Eclipse Modeling Framework* Second Edi., Amsterdam: Addison-Wesley Professional.

(Strang & Linnhoff-Popien, 2004)

Strang, T. & Linnhoff-Popien, C., 2004. A Context Modeling Survey. In *First International Workshop on Advanced Context Modelling, Reasoning And Management*. Nottingham, England, pp. 1–8.

(Su & Wu, 2011)

Su, C.-J. & Wu, C.-Y., 2011. JADE implemented mobile multi-agent based, distributed information platform for pervasive health care monitoring. *Applied Soft Computing*, 11(1), pp.315–325.

(Sun et al., 2010)

Sun, C., Rossing, R., Sinnema, M., Bulanov, P. & Aiello, M., 2010. Modeling and managing the variability of Web service-based systems. *Journal of Systems and Software*, 83(3), pp.502–516.

(Szyperski, 1998)

Szyperski, C., 1998. *Component Software: Beyond Object-Oriented Programming* 2nd ed. Addison-Wesley, ed., ACM Press.

(Tamura et al., 2014)

Tamura, G., Casallas, R., Cleve, A. & Duchien, L., 2014. QoS contract preservation through dynamic reconfiguration: A formal semantics approach. *Science of Computer Programming*, 94(P3), pp.307–332.

(The Apache Software Foundation, 2012)

The Apache Software Foundation, 2012. Apache Tuscany. Available at: http://tuscany.apache.org/ [Accessed July 7, 2016].

(The Eclipse Foundation, 2015)

The Eclipse Foundation, 2015. Papyrus Modeling Environment. Available at: http://www.eclipse.org/papyrus/ [Accessed August 13, 2016].

(Tran et al., 2009)

Tran, V.X., Tsuji, H. & Masuda, R., 2009. A new QoS ontology and its QoS-based ranking algorithm for Web services. *Simulation Modelling Practice and Theory*, 17(8), pp.1378–1398.

(UNEP, 2012)

UNEP, United Nations Environment Programme, 2012. *Early warning systems: State-of-art analysis and future directions*, Nairobi: United Nations Environment Programme.

(United Nations, 2001)

United Nations, 2001. *World Population Ageing: 1950-2050*, New York, USA.

(Vandewoude, 2007)

Vandewoude, Y., 2007. *Dynamically updating component-oriented systems.* Faculty of Engineering, K.U.Leuven, Leuven, Belgium.

(Vandewoude et al., 2007)

Vandewoude, Y., Ebraert, P., Berbers, Y. & D'Hondt, T., 2007. Tranquility: A Low Disruptive Alternative to Quiescence for Ensuring Safe Dynamic Updates. *IEEE Transactions on Software Engineering*, 33(12), pp.856–868.

(Vitabile et al., 2009)

Vitabile, S., Conti, V., Militello, C. & Sorbello, F., 2009. An extended JADE-S based framework for developing secure Multi-Agent Systems. *Computer Standards & Interfaces*, 31(5), pp.913–930.

(W3C, 2006)

W3C, 2006. Extensible Markup Language (XML) 1.1 (Second Edition). Available at: https://www.w3.org/TR/2006/REC-xml11-20060816/ [Accessed July 7, 2016].

(W3C, 2001)

W3C, 2001. Web Services Description Language (WSDL) 1.1. Available at: https://www.w3.org/TR/wsdl [Accessed September 6, 2016].

(Walderhaug et al., 2007)

Walderhaug, S., Stav, E. & Mikalsen, M., 2007. The MPOWER Tool Chain - Enabling Rapid Development of Standards-based and Interoperable Homecare Applications. In *Norsk Informatikk Konferanse*. Oslo, Norway, pp. 103–107.

(Wegdam et al., 2003)

Wegdam, M., Almeida, J.P.A., Sinderen, M.J. van & Nieuwenhuis, L.J.M., 2003. *Dynamic Reconfiguration for Middleware-Based Applications*. CTIT Technical Report TR-CTIT-03-09.

(Wehrmeister et al., 2014)

Wehrmeister, M.A., De Freitas, E.P., Binotto, A.P.D. & Pereira, C.E., 2014. Combining aspects and object-orientation in model-driven engineering for distributed industrial mechatronic systems. *Mechatronics*, 24(7), pp.844–865.

(Weiss, 1999)

Weiss, G., 1999. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence* G. Weiss, ed., Cambridge,Massachusetts: The MIT Press.

(Wooldridge & Jennings, 2009)

Wooldridge, M. & Jennings, N.R., 2009. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2), pp.115–152.

(World Health Organization, 2011)

World Health Organization, 2011. Global Health and Aging. Available at: http://www.who.int/ageing/publications/global_health/en/ [Accessed June 13, 2016].

(Wuyts & Ducasse, 2001)

Wuyts, R. & Ducasse, S., 2001. *Non-Functional Requirements in a Component Model for Embedded Systems*.

GLOSARIO

"No temas a las dificultades: lo mejor surge de ellas."

(Rita Levi–Montalcini)

GLOSARIO

AAL	*Ambient Assisted Living*
API	Interfaz de programación de aplicaciones (*Application Programming Interface*)
CADAMM	Context-Aware Distributed Applications Meta-Model
CBSE	Ingeniería del software basada en componentes (*Component-Based Software Engineering*)
DDS	*Data Distribution Service*
DRS	*Dynamic Reconfiguration Service*
ECA	*Event-Condition-Action*
EMF	*Eclipse Modeling Framework*
EMP	*Eclipse Modeling Project*
EPL	*Event Processing Language*
FSM	*Finite State Machine*
GCIS	Grupo de Control e Integración de Sistemas
GMF	*Graphical Modeling Framework*
GRAV	Grupo de Robótica, Automática y Visión por Computador
HW	*Hardware*

IDE	Entorno de desarrollo integrado (*Integrated Development Environment*)
IDL	Lenguaje de descripción de interfaces (*Inteface Description Language*)
IoT	Internet de las cosas (*Internet of Things*)
iLAND	*mIddLewAre for deterministic dynamically reconfigurable NetworkeD embedded systems*
JADE	*Java Agent Development*
FIPA	*Foundation for Intelligent Physical Agents*
FPS	Imágenes Por Segundo (*Frames Per Second*)
FSM	Máquina de estados finitos (*Finite State Machine*)
MAS	Sistema multi-agente (*Multi-Agent System*)
MDE	Ingeniería Dirigida por Modelos (*Model-Driven Engineering*)
M2M	Modelo a Modelo (*Model To Model*)
M2T	Modelo a Texto (*Model To Text*)
NFR	*Non-Functional Requierment*
NTP	*Network Time Protocol*
OASIS	*Organization for the Advancement of Structured Information Standards*
PKI	Infraestructuras de clave pública (*Public Key Infrastructure*)
QoS	Calidad de Servicio (*Quality of Service*)

SCA	*Service Component Architecture*
SOA	Arquitectura orientada a Servicio (*Service Oriented Architecture*)
SOC	Computación orientada a servicios (*Service Oriented Computing*)
SSL	*Secure Socket Layer*
SW	*Software*
UML	*Unified Modeling Language*
VPN	Red privada virtual (*Virtual Private Network*)
WSDL	Lenguaje de descripción de servicios web (*Web Service Description Language*)
XMI	*XML Metadata Interchange*
XML	*eXtensible Markup Lenguaje*
XSLT	*eXtensible Stylesheet Language Transformation*