Francisco Maciá Pérez
Virgilio Gilart Iglesias
Diego Marcos Jorquera
José Vicente Berná Martínez
Francisco José Mora Gimeno
Juan Antonio Gil Martínez-Abarca
Héctor Ramos Morillo
Joan Carles Monllor Pérez
Luis Felipe Herrera Quintero
Jorge Selva Soler
Jorge Gea Martínez
Antonio Ferrándiz Colmeiro
Iren Lorenzo Fonseca (Eds.)

Desarrollo de Grandes Aplicaciones de Red

VI Jornadas, JDARE 2009
Alicante, España, octubre 15-16, 2009
Actas

Grupo de Middleware
Departamento de Tecnología Informática y Computación
Universidad de Alicante

Título: Desarrollo de Grandes Aplicaciones de Red. VI Jornadas, JDARE 2009. Alicante, España, octubre 15-16, 2009. Actas

Editores: Maciá Pérez, Francisco; Gilart Iglesias, Virgilio; Marcos Jorquera, Diego; Berná Martínez, José Vicente; Mora Gimeno, Francisco José; Gil Martínez-Abarca, Juan Antonio; Ramos Morillo, Héctor; Monllor Pérez, Joan Carles; Herrera Quintero, Luis Felipe; Selva Soler, Jorge; Gea Martínez, Jorge; Ferrándiz Colmeiro, Antonio; Lorenzo Fonseca, Iren

Autores: Abreu Ortega, Miguel; Acuña Castillo, Silvia T.; Alfonso Aguilar, José; Almeida Cruz, Yudivián; Alpuente Hermosilla, Jesús; André Ampuero, Margarita; Baldoquín de la Peña, María G.; Barros Bastante, Laura; Báez Fildó, Ernesto; Barambones, Óscar; Berná Martínez, José Vicente; Bueno Rojas, Marisel; Cuenca Asensi, Sergio; Cuevas Cuesta, César; Del Barrio Fernández, Ángela; Díaz Pando, Numberto; Drake Moyano, José M.; Fernández Baizán, Covadonga; Fernández Oliva, Alberto; Ferrándiz Colmeiro, Antonio; Galiana Lara, Juan; Garea Llano, Eduardo; Germán, Ernesto; Gea Martínez, Jorge; Gil Martínez-Abarca, Juan Antonio; Gilart Iglesias, Virgilio; González Castaño, Idis; Herrera Quintero, Luis Felipe; Katrib Mora, Miguel; Lau Fernández, Rogelio; Lavastida López Zidia; López Espí, Pablo Luís; López Martínez Patricia; López Paz, Carlos Ramón; Lorenzo Fonseca, Iren; Maciá Pérez, Francisco; Marcos Jorquera, Diego; Martín Rodríguez Diana; Martínez Martínez, Carlos; Martínez Rojas, Juan Antonio; Mas Fernández, Ildefonso; Mediavilla Sánchez, Ángel; Mora Gimeno, Francisco José; Morales Vega, Daimí; Núñez Musa, Yulier; Oliva Santos, Rafael; Pastrana Cuanda, Léster; Quintana Pacheco, Yuri; Ramos Morillo, Héctor; Rosete Suárez Alejandro; Ruiz Fernández, Daniel; Sánchez Montero, Rocío; Sánchez Díaz, Léster; Sepúlveda Lima, Roberto; Somoza Moreno, Alfredo; Tamayo Castillo, Alejandro; Torres Pérez, Isis; Vera Voronisky, Francisco; Wilford Rivera, Ingrid; Zamanillo Sainz de la Maza, Isabel; Zamanillo Sainz de la Maza, José María

ISSN: 1889-7819
ISBN: 978-84-613-4894-7
Depósito legal: MU 2334-2009

Edita: GrupoM, Universidad de Alicante
www.dtic.ua.es/grupom
Carretera San Vicente - Alicante s/n
03690, San Vicente (Alicante)
(+34) 96 590 3681

Printed in Spain
Imprime: OFP Yeclagrafic

Aplicación de las tecnologías de componentes al desarrollo e integración de sistemas heterogéneos distribuidos y abiertos[1]

Laura Barros Bastante, Ángela del Barrio Fernández, Patricia López Martínez, César Cuevas Cuesta, José M. Drake Moyano

Avda. Castros s/n, 39005 Santander -Spain
{barrosl, delbarrioa, lopezpa, cuevasce, drakej}@unican.es
http://www.ctr.unican.es

Resumen. En este trabajo se describe la aplicación de las tecnologías de componentes para la integración de aplicaciones y servicios que se despliegan en una plataforma distribuida, heterogénea y abierta. Se ha utilizado como base el modelo de componentes CCM y la especificación de configuración y despliegue D&C, ambos propuestos por OMG. De ellos se mantiene la metodología de especificación, el modelo de referencia y el proceso de desarrollo, pero se ha eliminado la dependencia de CORBA y se ha introducido la capacidad de usar varios middlewares y servicios de comunicaciones simultáneamente. La estrategia de integración se basa en considerar al componente como la unidad con la que se diseñan y despliegan las aplicaciones. Los componentes mantienen sus características de aislamiento, opacidad y reusabilidad y se distribuyen como paquetes que incluyen conjuntamente el código y los metadatos necesarios para que con la ayuda de herramientas automáticas puedan ser integrados en las aplicaciones y desplegados sobre diferentes plataformas de ejecución.

1 Ámbito de aplicación

Actualmente es muy frecuente que en instalaciones industriales o de servicios se utilice una plataforma distribuida basada en *ethernet* con computadores que se instalan sobre un área geográfica extensa y que dan soporte a una gran variedad de aplicaciones de supervisión y control de las instalaciones y actividades que se producen en ella. Se elige esta arquitectura descentralizada porque minimiza el coste de la infraestructura y es altamente escalable. El sistema suele ser muy heterogéneo ya que los computadores son frecuentemente parte de equipos instalados o utilizan periféricos específicos que conllevan el uso de procesadores, sistemas operativos y protocolos de comunicaciones muy diversos. Así mismo, las aplicaciones a las que dan soporte son de naturaleza muy variada, y han sido desarrolladas por diferentes empresas y en diferentes etapas de instalación. Sin embargo, son interdependientes porque hacen uso de múltiples servicios comunes.

La tecnología que se presenta en el presente artículo, se ha realizado en el ámbito del proyecto de investigación HESPERIA [1], en el que un conjunto de más de 20 empresas y grupos de investigación han desarrollado e integrado tecnologías y

[1]Este trabajo ha sido realizado como parte del proyecto HESPERIA [1] financiado por el programa español de Consorcios Estratégicos Nacionales en Investigación Técnica (CENIT 2005).

productos destinados a sistemas de seguridad, vídeo-vigilancia y control de operaciones en infraestructuras de espacios públicos —aeropuertos, áreas comerciales, museos, etc.— y estratégicos —centrales nucleares, presas, tendidos ferroviarios, etc.—. En los sistemas para los que se propone la estrategia, se pueden identificar como relevantes las siguientes características:

- Orientados a arquitecturas cliente/servidor de múltiples capas. La arquitectura cliente/servidor proporciona un modelo de referencia idóneo a sistemas en los que coexisten múltiples aplicaciones distribuidas complejas que interaccionan entre sí a través de los servicios que comparten. Así mismo, las arquitecturas de múltiples capas conducen a clientes ligeros, servidores intermedios que soportan la funcionalidad y servidores terminales que implementan los servicios de la infraestructura que son comunes a diferentes aplicaciones.
- Plataforma altamente heterogénea que utiliza simultáneamente procesadores, sistemas operativos y redes de comunicación de tipos muy diferentes. Las aplicaciones y servidores han sido desarrollados en diferentes lenguajes de programación y muchos de ellos son productos legados opacos.
- Disponen de algún *middleware* que aísla las aplicaciones de las características específicas de los elementos de la plataforma, y que resuelve la comunicación entre módulos codificados en diferentes lenguajes y ejecutados en diferentes sistemas operativos.
- Alta escalabilidad. En este ámbito es frecuente tener instalaciones cuya complejidad pueda necesitar ser escalada en varios órdenes de magnitud.
- Fiables y robustos frente a los fallos de las aplicaciones que se ejecutan. Deben garantizar la operatividad de cada aplicación frente a fallos de otras, detectar las aplicaciones o servicios con fallos, y reconfigurar “en vivo” las aplicaciones con fallos para mantener su operatividad.
- Fiables y robustos frente a sobrecargas mediante servicios de planificación de los recursos compartidos que garantizan su disponibilidad a fin de que las aplicaciones satisfagan sus requisitos de comportamiento.
- Capacidad de gestión centralizada y local del ciclo de vida de las aplicaciones y de los servidores del sistema.

En este ámbito, las propuestas tecnológicas y arquitecturales para sistemas distribuidos heterogéneos han evolucionado rápidamente en los últimos diez años. En la figura 1, se muestra la evolución de las tecnologías promovidas por OMG:

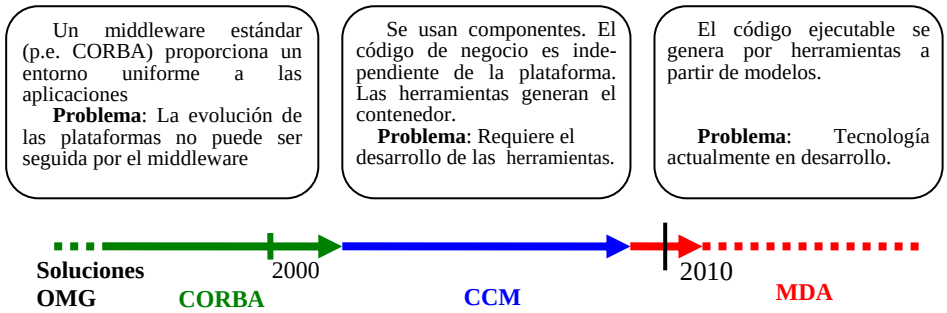

Fig. 1. Evolución de las tecnologías de sistemas distribuidos heterogéneos.

- Solución por el *middleware*. A finales de los años 90, la solución se buscaba a través de un *middleware* universal — como CORBA— que servía de base y nexo de todas las aplicaciones que se ejecutaban en la plataforma distribuida. Esta solución cayó en declive porque los elementos de la plataforma evolucionaban tan rápidamente que no podía ser seguido por las implementaciones del *middleware*. En el año 2002, OMG publica la última revisión principal 2.5 de CORBA [2] y luego renuncia a seguir esta vía.
- Tecnología de componentes. OMG inicia una nueva vía con la publicación de la especificación de componentes CCM [3]; en ella se plantea que los módulos distribuibles (componentes) se elaboran en dos fases. El código de negocio lo desarrolla el experto del dominio de aplicación, sin necesidad de conocer la plataforma, mientras que el contenedor que lo adapta a la plataforma — o al *middleware*— es desarrollado por el experto de plataforma y habitualmente automatizado a través de herramientas. Aunque puede considerarse que es la tecnología vigente, su declive ya es evidente, porque de nuevo el dinamismo de la plataforma exige un esfuerzo excesivo en el desarrollo de herramientas.
- Estrategias MDA/MDE. La solución que actualmente ofrece OMG se basa en el desarrollo de meta-herramientas que, en función de los modelos de los elementos de las plataformas generan las herramientas generadoras del código que se despliega en ellas.

La tecnología que se describe en este artículo se corresponde a una fase de evolución comprendida entre la segunda y la tercera opción. Se utiliza la especificación y el modelo de referencia CCM propio de la tecnología de componentes, pero utilizando metodologías MDA para crear las herramientas de generación de los contenedores.

2 Componentes basados en contenedor

Frente a la designación original de OMG de CCM como *CORBA Component Model*, en recientes proyectos europeos como COMPARE [4] o FRESCOR [5], se interpreta la tecnología en el sentido de *Container Component Model*, esto es, una tecnología para el desarrollo de aplicaciones distribuidas que utiliza el modelo de referencia LwCCM [6], eliminando la dependencia de CORBA como *middleware* y que utiliza la especificación D&C [7] de OMG para estandarizar la configuración y despliegue de las aplicaciones. Siguiendo este modelo de referencia, el código de negocio del componente se desarrolla independientemente de la tecnología subyacente, siendo el contenedor del componente el que ofrece todos los recursos necesarios para adaptar dicho código a la plataforma y *middleware* que se esté utilizando. Dicho contenedor es generado de forma automática a partir de los metadatos que se asocian a la descripción de los componentes. Esta tecnología se viene usando en sistemas en los que predomina la heterogeneidad, la naturaleza embebida de muchos de sus elementos o cuando presentan requisitos no funcionales y de tiempo real.

Ejemplos de tecnologías que han sido desarrolladas siguiendo esta estrategia, son:

- MicroCCM [8], destinada a sistemas embebidos y basada en un *middleware* CORBA ligero propio, desarrollada por la empresa Thales France.

- Ada-CCM [9] destinada a sistemas embebidos de tiempo real y que se comunican a través de *sockets* con protocolo RTEP [10] y bus CAN, desarrollada dentro del proyecto del plan nacional THREAD [11].
- hCCM, basada en la adaptación de Ice, un *middleware* orientado a objetos distribuidos y que permite plataformas con procesadores Windows, Unix/Linux y JVM, y que admite software codificado en Java y C/C++.

En esta última tecnología, que es la que se presenta en este artículo, el intercambio de flujo de vídeo o de sonido en tiempo real entre componentes, requiere que se utilice tanto la interacción basada en el *middleware* de base Ice, como comunicaciones basadas en conectores que encapsulan mecanismos de comunicación con otros tipos de protocolos tales como RTP [12].

Además de los principios de aislamiento, reusabilidad y opacidad que son propios del paradigma de componentes, un objetivo específico de la tecnología hCCM es independizar el diseño y desarrollo del código de negocio de los componentes de las características de la plataforma en que se vaya a ejecutar. De esta forma, un componente es un módulo software que implementa una determinada funcionalidad y que puede ser desarrollado por un experto del dominio de aplicación que no necesita conocer, ni utilizar, elementos específicos del *middleware* o de la plataforma en que se vaya a ejecutar. Satisfecho este objetivo, el trabajo del desarrollador de componentes se hace más confortable, ya que puede elegir el lenguaje de programación que le sea mas cómodo, realizar su trabajo como si estuviese en un entorno monoprocesador, y confiar que el código que resulta de su trabajo será, en cada caso, adaptado a la plataforma concreta en que se vaya a ejecutar por expertos del *middleware*. Estos a su vez tratarán el código de negocio como un producto opaco cuya estructura y detalles internos no necesitan conocer.

En la figura 2, se muestran los elementos que constituyen un componente ejecutable con las características mencionadas:

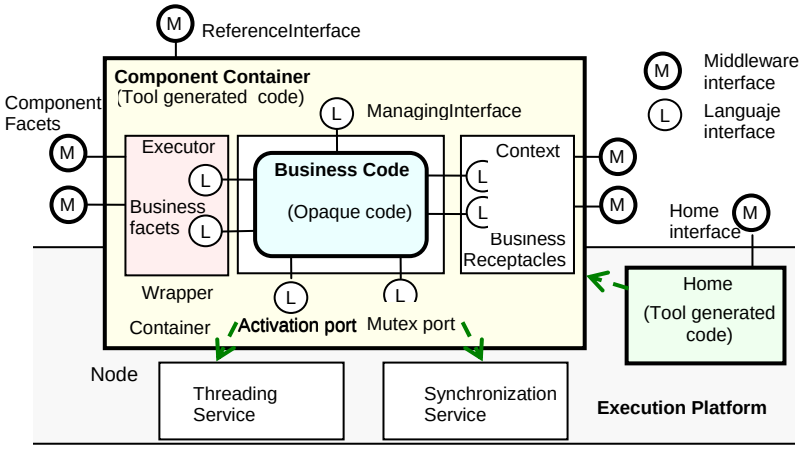

Fig. 2. Principales elementos que constituyen un componente ejecutable.

El código de negocio es el que implementa la funcionalidad del componente. Su estructura interna puede ser establecida con absoluta libertad por el diseñador. Sin embargo los puertos que debe ofrecer externamente están establecidos por la tecnología, aunque las interfaces que implementan están formuladas en el lenguaje de

programación que se haya elegido. Como ayuda, todas esas interfaces pueden generarse automáticamente a partir de la especificación del componente. Los puertos a implementar o usar por el código de negocio se pueden clasificar en siete tipos:

- Puerto de referencia: establece las operaciones que debe ofrecer para gestionar su ciclo de vida, establecer su configuración, proporcionar acceso a sus puertos y conectar sus receptáculos.
- Facetas: puertos por lo que los clientes acceden a la funcionalidad ofrecida por el componente.
- Receptáculos: puertos por los que el componente obtiene de otros componentes la funcionalidad externa que requiere para implementar la funcionalidad que ofrece.
- Receptáculos de emisión de eventos: requieren el puerto del gestor de eventos por el que el componente los transfiere hacia los componentes que se han declarado interesados por ellos.
- Facetas de gestión de eventos: puertos que definen las operaciones ejecutadas en respuesta a los eventos que se reciben del servicio de gestión de eventos.
- Puertos de activación: puertos que definen la actividad que deben ejecutar los *threads* del entorno requeridos por el componente para implementar la funcionalidad interna.
- Receptáculos de sincronización: requieren la gestión de los *mutex* y variables de condición proporcionados por el entorno. El componente los necesita para sincronizar los múltiples *threads* que ejecutan concurrentemente su código.

El contenedor representa el código que adapta el código de negocio del componente al *middleware* y demás servicios de la plataforma de ejecución. El contenedor no forma parte del componente cuando se transfiere desde el programador o empresa que lo ha desarrollado al ensamblador o empresa que lo utiliza en una determinada aplicación. El código del contenedor se genera en la fase de despliegue de las aplicaciones mediante herramientas de generación de código que encierran en su diseño el conocimiento experto del *middleware* y de los recursos de la plataforma de ejecución, así como las reglas de la tecnología de componentes.

La estructura interna del contenedor, así como los puertos y las interfaces que implementa están definidos en la especificación CCM y ha sido propuesta con el objetivo de simplificar la generación automática de su código mediante herramientas. Los elementos básicos que incluye, son:

- El contenedor propiamente dicho (*wrapper)* que crea y aloja los demás elementos y ofrece la interfaz de referencia a través de la que se gestiona el componente y se accede a sus puertos.
- El ejecutor (*executor*), que ofrece las facetas del componente e implementa los *servant* a través de los que se invocan las facetas del código de negocio.
- El contexto (*context*), que contiene los *proxys* a través de los que se accede a las facetas de aquellos componentes que se conectan a sus receptáculos.

La implementación de los componentes requiere frecuentemente *threads* internos que o bien ejecutan actividades en respuesta a eventos hardware procedentes del entorno, de los servicios de comunicación o del reloj del sistema, o se requieren por la estrategia de diseño con la que se implementa, cuando se utilizan respuestas asíncronas, tareas periódicas de escrutinio, mecanismos de arbitrio activo, etc. Sin

embargo, mientras que en ciertos lenguajes como Java o Ada, los *threads* pueden gestionarse desde el lenguaje de programación, en otros muchos como es el caso de C/C++, es el sistema operativo el que los crea y aplica. A fin de que los componentes software sean independientes de estas alternativas, se ha optado porque el código de negocio sea siempre código pasivo, y cuando por razones de diseño se requiere disponer de concurrencia interna, requiera los *threads* del entorno a través de los puertos de activación que ya han sido mencionados. Cuando un componente requiere *threads*, el contenedor utiliza los servicios del gestor de *threads* (*Threading Service)* que a través de una interfaz definida en la tecnología crea y proporciona el *thread*.

La ejecución del código de un componente es habitualmente concurrente. Lo ejecutan tanto los *threads* internos, como aquellos otros que proceden de clientes que ejecutan servicios del componente. A fin de que la ejecución concurrente y los accesos a los recursos protegidos que gestiona el componente sean seguros, los componentes necesitan disponer de mecanismos de sincronización tales como *mutexes* o variables de condición. Como la creación y gestión de estos elementos son a veces realizadas desde el lenguaje, pero en otros muchos casos suplidos desde el sistema operativo, se ha optado por que siempre sea suplidos desde el contenedor a través de receptáculos de sincronización definidos por la tecnología. De acuerdo con la naturaleza del entorno de ejecución, y del lenguaje de programación, o bien el contenedor los genera y los suple al código de negocio del componente, o los requiere del entorno a través del servicio de recursos de sincronización (*Synchronization Service*) que los crea y suministra. En la figura 3, se muestran las interfaces definidas por la tecnología para la gestión de los threads y mecanismos de sincronización.

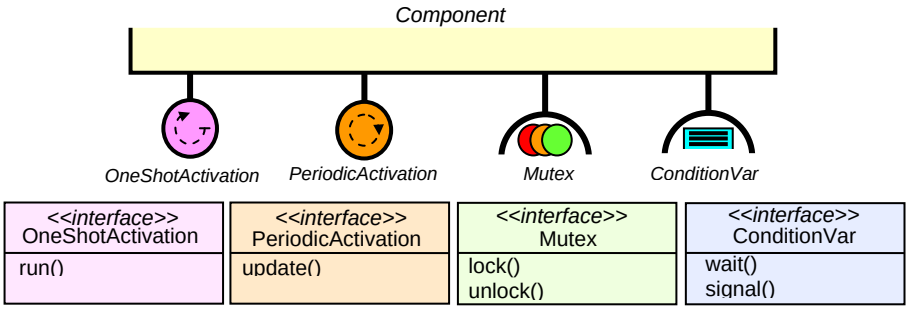

Fig. 3. Interfaces de gestión y sincronización de *threads* en hCCM.

La especificación CCM utiliza un mecanismo estático (*Home)* para configurar, crear, localizar y eliminar las instancias de cada tipo de componente que se instancia en un nudo procesador.

En la figura 4, se describen las fases del proceso con el que se desarrolla un componente como módulo software comerciable y distribuible independientemente, y que puede ser utilizado por otros para ensamblar futuras aplicaciones.

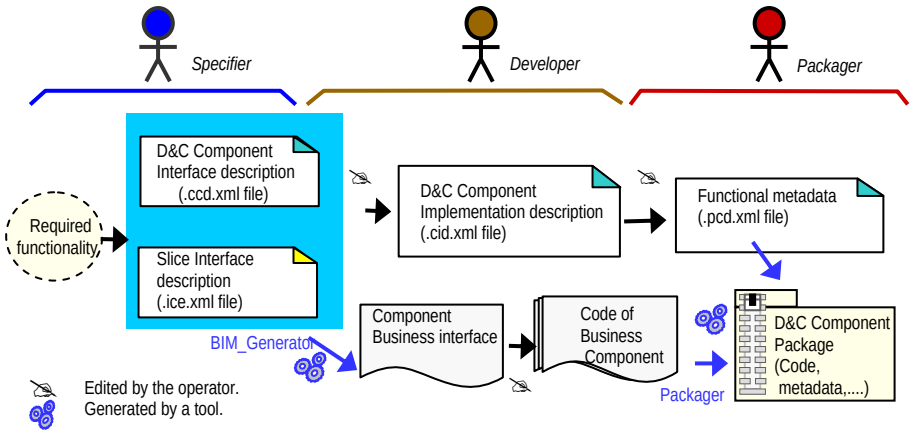

Fig. 4. Proceso de desarrollo de un componente

El proceso se realiza en tres fases y en cada una de ellas interviene un agente que es experto en un aspecto diferente del componente.

El especificador (*specifier*) es el experto en el dominio de aplicación, que cuando detecta una funcionalidad no cubierta, propone la especificación de un nuevo componente. La especificación de un componente se realiza a través de la interfaz del componente que representa su vista externa, es común a todas las implementaciones que se hagan del componente y contiene toda la información que se necesita para decidir sobre la utilidad del componente en una determinada aplicación. Básicamente describe la funcionalidad que ofrece, a través de la declaración de sus facetas, la funcionalidad que requiere, a través de la declaración de sus receptáculos, y las opciones de configurabilidad que posee, a través de la declaración de los parámetros de configuración que tiene definidos. El estándar D&C de OMG, establece el formato y el contenido del documento de tipo *ComponentInterfazDescription* —fichero de extensión *.ccd.xml*— con el que se describen las interfaces de los componentes.

El desarrollador (*Developer*) es el experto en diseño y codificación de software. Realiza el diseño y escribe el código de negocio del componente, utilizando únicamente como referencia la especificación del componente formulada por el especificador y las reglas establecidas en la tecnología. Su resultado son un conjunto de ficheros de código, así como los recursos que se requieren de la plataforma en los que la implementación pueda ser instanciada. El estándar D&C define el formato y el contenido del fichero de tipo *ComponentImplementationDescription* —fichero con extensión *.cid.xml*— que describe las implementaciones de los componentes.

El empaquetador (*packager*) es el experto en la tecnología que construye el paquete con el que se distribuyen y comercializan los componentes como entes autocontenidos y reutilizables. El estándar D&C define el formato y el contenido del fichero *PackageConfiguration* —fichero con extensión *.pcd.xml*— con la información que describe la única interfaz del componente, las múltiples implementaciones que se hayan realizado y al propio fichero —*.jar* o *.zip*— con el que se distribuye. Este paquete, que contiene código y metadatos constituye propiamente el componente como producto que puede ser utilizado de forma independiente.

3 Plataforma distribuida, heterogénea y abierta

En el estado actual de la tecnología de computadores, cuando se aborda la instrumentación y control de una instalación que ocupa una amplia zona geográfica, como, por ejemplo un aeropuerto o un museo, el mínimo costo se obtiene si por todo él se despliega una red de comunicaciones de bajo costo —por ejemplo WIFI o ethernet— y se introducen un gran número de computadores que están situados muy cerca de los sensores, e incluso integrados en ellos —en las cámaras, en los micrófonos, en los lectores de los puntos de acceso, etc.—. En este tipo de procesadores embebidos hay una gran limitación de recursos y sólo se pueden soportar *middlewares* muy ligeros. Por su costo y flexibilidad muchos de los procesadores tienen actualmente una arquitectura interna tipo PC, y sus sistemas operativos son Linux o Windows. También suelen existir servidores —por ejemplo, los que soportan las bases de datos— que tienen arquitecturas hardware más sofisticadas, pero en estos casos, también es muy frecuente que operen sobre Linux. Así mismo, las estaciones de monitorización y las interfaces de usuario utilizan muy frecuentemente la tecnología Java. En la versión actual de la tecnología hCCM que se presenta en este artículo, se ha considerado que los sistemas operativos únicamente sean algunas implementaciones de Linux y Windows NT y XP.

El diseño y despliegue de una aplicación informática en una plataforma distribuida de esta naturaleza es compleja, ya que requiere desplegar muchas secciones o módulos por los diferentes procesadores de la plataforma y mantener entre ellos una colaboración intensa. Así mismo, en la plataforma se van a estar ejecutando concurrentemente muchas aplicaciones diferentes, y hay que garantizar el acceso seguro a todos los recursos y servicios que comparten y tener la capacidad de desplegar y eliminar las aplicaciones "en vivo" sin afectar al estado de las restantes aplicaciones en ejecución. Todas estas situaciones requieren el uso intensivo de los recursos y servicios de los sistemas operativos. La necesidad de conocer y ser experto simultáneamente del dominio de las aplicaciones y de la plataforma —sistemas operativos, comunicaciones, etc.— dificulta considerablemente el desarrollo de las aplicaciones distribuidas en estos entornos. La solución a este problema es introducir un *middleware* como elemento homogenizador de la diversidad que existe.

En el proyecto Hesperia, se tomó la decisión de usar el *middleware* ICE (*Internet Communications Engine*) [13], que está orientado a objetos distribuidos, y que es mucho más ligero que CORBA. Soporta múltiples lenguajes de programación y es compatible con los sistemas operativos que se van a utilizar y con las limitaciones de recursos de los procesadores embebidos. ICE es software libre y está ofrecido bajo los términos de Licencia Pública General de GNU (GPL).

El hecho de que ICE sea un *middleware* de un único suministrador y que no siga ningún estándar, hace más relevante la estrategia seguida en hCCM de desacoplar radicalmente el código de negocio del *middleware* y de sus servicios. Si en el futuro hay que migrar a un nuevo *middleware*, sólo hay que cambiar las herramientas de generación de los contenedores y no se necesita modificar el código de negocio.

La tecnología de componentes hCCM no sólo simplifica el desarrollo de las aplicaciones al independizar el desarrollo del código de negocio de la plataforma, sino que también, a través del uso de conectores, permite introducir mecanismos de comunicación especiales que no puedan ser ofertados por el *middleware* —p.e.

transmisión de vídeo y sonido continuo en tiempo real—. Como se muestra en la figura 5, el conector es un elemento especializado compuesto por dos componentes estándar emparejados, que encapsula en su código interno, el mecanismo de comunicación que se necesita para establecer la interacción entre un cliente y un servidor por el medio de comunicación que se necesita.

La funcionalidad que debe ser implementada por un conector es la serialización y deserialización de la información que intercambia, la transferencia de las invocaciones a través de la red, la sincronización del cliente a la espera o no de que la invocación termine, y la invocación local del servicio en el componente servidor. Todas estas tareas son muy uniformes, lo que facilita que el código de los conectores se pueda generar en la fase de despliegue haciendo uso de herramientas de generación de código.

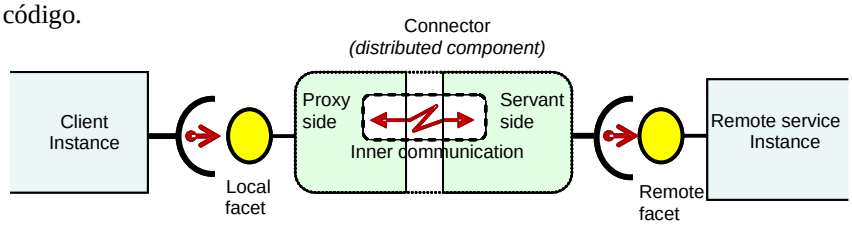

Fig. 5. Elementos de un conector que realiza la interacción entre dos componentes

Un problema que hay que resolver en las aplicaciones distribuidas es el mecanismo a través del que los diferentes componentes que constituyen una aplicación se localizan y obtienen la información de acceso para establecer la comunicación entre ellos. En hCCM estos mecanismos se introducen como parte del contenedor, y por tanto son transparentes al código de negocio. En la versión actual se dispone de tres mecanismos:

- Conexión estática basada en direcciones de comunicación conocidas. Se puede aplicar cuando todos los componentes que constituyen la aplicación se despliegan simultáneamente. En ese caso se conocen las direcciones de acceso a cada puerto de los componentes, y solo es necesario intercambiarlas entre ellos para establecer la conexión.
- Conexión dinámica basada en servidores de nombres. Los servidores que pertenecen a la infraestructura, se instalan con independencia de las aplicaciones. Para su localización se suelen utilizar los servicios de nombres que son ofrecidos por el *middleware*. Cuando se instalan los servidores se registran en el servicio de nombres con una clave, y posteriormente cuando se despliegan las aplicaciones, los clientes reciben las claves como parámetros de configuración y a través del servicio de nombres obtienen las direcciones de acceso a los servicios de infraestructura.
- Conexión dinámica basada en encuesta y respuesta: cuando los sistemas se hacen muy complejos y dinámicos, la conexión basada en servicios de nombres y claves se hace muy difícil de gestionar. Para estos casos se ha establecido un mecanismo conocido como ASDF (*Abstract Service Discovery Framework*) [14] basado en el requerimiento de componentes que satisfacen cierto conjunto de propiedades y respuesta de aquellos que las satisfacen. Una ventaja de este

mecanismo es su sencillez, y el hecho de que sólo afecta a los componentes de una misma aplicación.

En la tecnología hCCM, la plataforma de ejecución se encuentra descrita mediante un modelo que describe los nudos de procesamiento y las redes de comunicación que existen en ella, así como los recursos que ofrecen cada uno de estos elementos. Esta información es utilizada por las herramientas de despliegue para asignar a la instancias de los componentes el nudo en el que se instalan, y para elegir, de entre las múltiples implementaciones del componente, aquella que es mas idónea, en función del balance entre recursos requeridos por la implementación para instalarse, y los recursos de que dispone nudo de procesamiento asignado. El estándar D&C de OMG especifica el formato y la información del documento *TargetDataModel* —fichero con extensión *.cim.xml*— que describe la plataforma de ejecución.

4 Configuración y despliegue de servicios y aplicaciones

En la tecnología hCCM el componente es la unidad básica de despliegue. Tanto el código de negocio de los componentes como los metadatos que describen su funcionalidad y la información para su gestión deben estar disponibles en el repositorio del entorno de desarrollo de aplicaciones. Cuando se despliega una aplicación se hace uso de un conjunto de herramientas disponibles en el entorno, se generan los códigos ejecutables y se lanza su ejecución en el nudo procesador que se le haya asignado. Una aplicación despegable es simplemente un componente más, cuyo despliegue tiene interés de por si. A veces es un conjunto de componentes agrupados como un componente compuesto (*Component Assembly*) posiblemente diseñado específicamente para su despliegue, y otras veces es un componente simple (*Monolithic Implementation*) que ofrece una funcionalidad de la que interesa disponer en el sistema. También se pueden desplegar de forma individual los servicios de infraestructura, que también son componentes —*monolithics* o *assemblies*— cuya función es suplir servicios a aplicaciones que serán desplegadas más adelante.

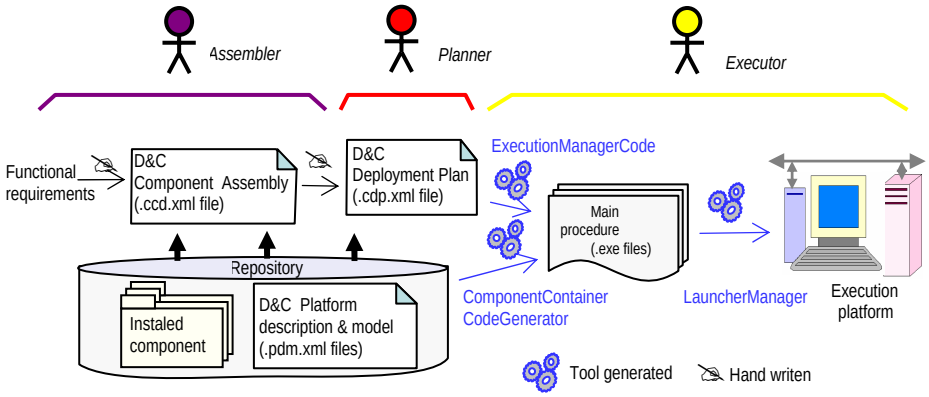

Fig. 6. Proceso de desarrollo de una aplicación en hCCM.

En la figura 6, se muestra el proceso de despliegue de una aplicación o servicio. El ensamblador (*assembler*) elabora el componente compuesto o selecciona el componente simple que va a desplegarse. Así mismo, de acuerdo con la funcionalidad requerida asigna los valores de configuración que corresponden a cada instancia de componente que participa en la aplicación. El planificador (*planner*) toma las decisiones de despliegue, esto es, asigna a cada instancia el procesador en que se va a instalar y decide el mecanismo de comunicación entre las instancias de componente. Todo ello lo codifica en un documento denominado plan de despliegue (*Deployment Plan*) — fichero con extensión *.cpd.xml*—, que contiene la información completa que se necesita para el despliegue de la aplicación. Su formato y contenido están especificados en el estándar D&C de OMG. Por último, el ejecutor (*Executor*) hace uso de las herramientas de despliegue: genera el código ejecutable de cada componente usando la herramienta *ComponentContainerCodeGenerator* y genera el código de ejecución final para cada nudo con la herramienta *ExecutionManagerCode.*

5 Ejemplo de aplicación

En este apartado, a través de un ejemplo sencillo, se muestran las posibilidades de reutilización y despliegue de la estrategia que se presenta. El ejemplo es la aplicación HomelandAlarm cuya función es la detección de situaciones de alarma, y la generación de respuestas programadas cuando se pasa de estado normal a estado de alarma o viceversa. Como se muestra en la figura 7, está organizada en tres capas:

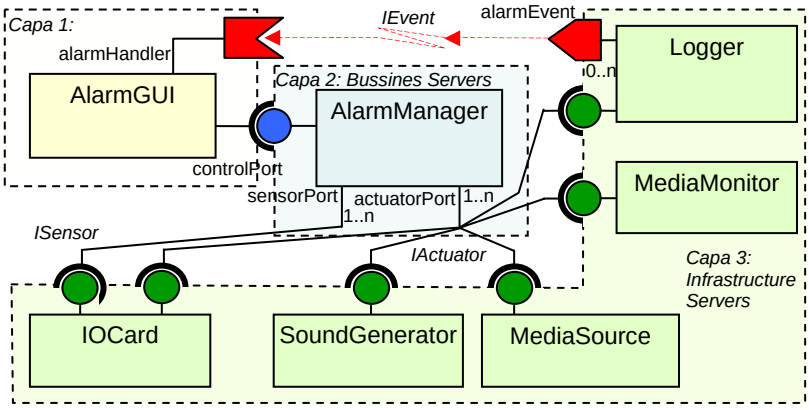

Fig. 7. Arquitectura software de la aplicación HomelandAlarm.

- Capa 1. *Clients*: los clientes suelen ser interfaces gráficas de usuario ligeras que actúan como elemento de interacción con los operadores. Habitualmente, son componentes desarrollados específicamente para cada aplicación. Por ejemplo, si este sistema se aplicara al control de los elementos de confort de un edificio — accesos, calefacción, control de energía, etc. —, o al control de un parking —vigilancia, control de plazas, etc.—, sólo los componentes de la capa 1 necesitan ser desarrollados específicamente para cada caso.

- Capa 2. *Business Servers*: son componentes complejos en los que reside la funcionalidad de la aplicación. Se han diseñado para que sean reutilizables. En los ejemplos de aplicación mencionados de control del confort de un edificio o de un parking, se utiliza el mismo componente. La configuración que habría que establecer sería distinta en una aplicación u otra.
- Capa 3. *Infrastructure Servers*: está constituida por los componentes ligados a la infraestructura instalada sobre la zona, que son compartidos por esta aplicación y por muchas otras que se están ejecutando. En el ejemplo, algunos componentes de esta capa implementan la interfaz ISensor y a través de ellos se puede leer el estado de algún tipo de alarma, así, un elemento de este tipo es el componente IOCard que permite leer señales analógicas —temperaturas, nivel de CO_2, etc.— o digitales —estado de una puerta, presencia de un coche, etc. —. Otros componentes implementan la interfaz IActuator, mediante ellos se pueden realizar acciones de respuesta: establecer el estado de una línea digital (IOCard), generar un sonido (SoundGenerator), monitorizar una señal de vídeo en un monitor (MediaMonitor), registrar la imagen que se captura por una cámara (MediaSource), registrar un mensaje (Logger), etc.

El punto de partida del proceso de configuración y despliegue de una aplicación es la información disponible en el repositorio del entorno de despliegue. En él se encuentra registrada la información relativa a los componentes que se utilizan y a la plataforma distribuida en que se van a ejecutar. Por cada componente existe un paquete de información que contiene la descripción de la interfaz del componente — funcionalidad, interfaces que ofrece y requiere, opciones de configuración, etc. — y la descripción de las implementaciones de cada componente — código de negocio, requisitos de instanciación, mecanismo de ejecución, etc. —. El *Planner* elabora el Plan de Despliegue, que es una estructura de datos que describe la aplicación que se desarrolla: las instancias de los componentes que se ensamblan, la asignación de los procesadores en que se instancian y los mecanismos de comunicación con los que interaccionan, etc.. En los ejemplos de aplicación que se consideran, el plan de despliegue contiene la descripción de las instancias que se instalan: AlarmManager y AlarmGUI — ConfortGUI o ParkingGUI— según sea la aplicación que se despliega y las referencias a las instancias de los componentes de la infraestructura con los que se ensamblan.

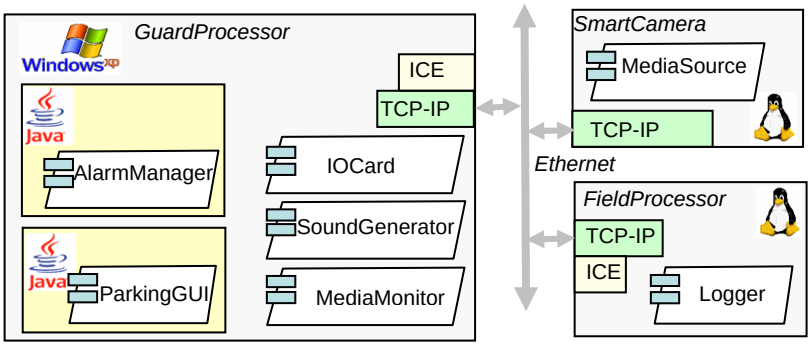

Fig. 8. Despliegue de la aplicación ParkingApp.

En la figura 8 se muestra una opción de despliegue de la aplicación ParkingApp. La configuración, despliegue y ejecución de la aplicación las realiza el ejecutor con las herramientas que son específicas de la tecnología que se utilizan en la plataforma — en este caso hCCM—. Conducida por el plan de despliegue, los metadatos disponibles en los componentes, y la descripción de la plataforma de ejecución:

- Genera y compila el código de los adaptadores y conectores, y lo enlaza con los códigos de negocio de los componentes hasta generar el código ejecutable que corresponde a cada nudo procesador.
- Gestiona la información de configuración de los componentes, que especializan las instancias de los componentes en función de los requisitos de la aplicación y proporciona los localizadores de las instancias de los componentes de la infraestructura con los que debe ensamblarse.
- Genera los ficheros — *jar*, *ddl*, *makefile*, etc.— compatibles con el sistema operativo de la plataforma, con los que se instala la aplicación en cada nudo.

Una de las ventajas de la estrategia que se presenta es la independencia de los códigos y de los metadatos de los componentes con la plataforma que se utiliza. Por ejemplo, si en vez de utilizarse el *middleware* ICE que se indica en la figura, se utilizara CORBA, nada habría que cambiar de la descripción de los componentes, ni del plan de despliegue, sólo habría que utilizar las herramientas de despliegue que son específicas para esa nueva tecnología.

6 Conclusiones

Se ha presentado una tecnología y una estrategia para la configuración y despliegue de aplicaciones que deben ejecutarse en plataformas distribuidas complejas orientadas al control y supervisión de grandes superficies geográficas y que se caracterizan por su heterogeneidad. Se utiliza como referencia el modelo de componente CCM que propone OMG, con algunas extensiones y cambios para evitar la dependencia de CORBA que incluye la especificación de OMG.

La principal contribución de la estrategia consiste en la total independencia del diseño del código de negocio de los módulos software respecto de la adaptación a la plataforma. El desarrollador de los componentes, de los servicios y de las aplicaciones no necesita conocer la plataforma ni el *middleware* que se utiliza en ella. Desarrolla el software como si trabajase en un entorno monoprocesador y con el lenguaje de programación que considera más adecuado a su dominio de aplicación. La adaptación a la plataforma distribuida concreta en la que se despliega lo realizan herramientas automáticas que incorporan en su funcionalidad el conocimiento del *middleware* y de la tecnología que se utiliza en la plataforma.

La gran ventaja de la estrategia es el incremento de la reusabilidad de los productos software que se utilizan. No sólo heredan de la tecnología de componentes la reusabilidad entre aplicaciones, sino que también son directamente reutilizables — esto es, sin necesidad de cambiar ni conocer el código— sobre plataformas basadas en diferentes *middlewares*.

Aunque no se ha tratado en este artículo, otra gran ventaja de la estrategia es la posibilidad de independizar los requisitos de comportamiento (performance) y de

tiempo real del código de negocio [15]. La gestión de los recursos para que se satisfagan los requisitos de respuesta temporal también se pueden incluir en el código de los contenedores, quedando la configuración para la planificabilidad de las aplicaciones bajo la responsabilidad del Planner y de las herramientas que éste utiliza.

Referencias

1. Proyecto HESPERIA: Homeland sEcurity: tecnologíaS Para la Seguridad integRal en espacIos públicos e infrAestructuras. Proyecto CENIT- 2005. https://www.proyecto-hesperia.org/.
2. OMG: Object Management Group,“Common Object Request Broker Architecture,” version 2.5, OMG document number formal/01-09-01.
3. OMG : "CORBA Components Model Specification", version 1.2, Formal/05-01-04.
4. IST projects: “COMPARE (Component-based approach for real-time and embedded systems)”. http://www.ist-compare.org.
5. IST project: “FRESCOR (Framework for Real-time Embedded Systems based on Contracts)”. http://www.frescor.org.
6. OMG: Lightweight Corba Component Model, ptc/03-11-03, November 2003.
7. OMG: Deployment and Configuration of Component-Based Distributed Applications Specification, version 4.0, Formal/06-04-02, April 2006.
8. THALES Land & Joint Systems, France: “Experience Report on Implementing and Applying a Standard Real-Time Embedded Component Platform”, OMG RTE Systems Workshop ,Washington, July 2007.
9. P.López, J.M. Drake, P. Pacheco, J.L. Medina: “An Ada 2005 Technology for Distributed and Real-Time Component-Based Applications” 13th International Conference on Reliable Software Technologies – Ada Europe (2008).
10. J.M. Martínez and M. González.: RT-EP: A Fixed-Priority Real Time Communication Protocol over Standard Ethernet In: Proc. of the 10th Int. Conference on Reliable Soft-ware Technologies, Ada-Europe 2005, York(UK), June 2005.
11. Proyecto THREAD(Soporte integral para sistemas empotrados de tiempo real distribuidos y abiertos). Plan nacional de investigación. Programa TIC (TIC 2005-08665-C03)
12. RTP: Real Time Protocol. RFC 1889 (http://tools.ietf.org/html/rfc1889).
13. M. Henning y M. Spruiell: “Distributed Programming with ICE”. http://www.zeroc.com.
14. D. Villa, F.J Villanueva, F. Moya, J. Barba, F. Rincón, J.C. López. Minimalist Object Oriented Service Discovery Protocol for Wireless Sensor Networks. Design Automation and Test in Europe (DATE'07). 2007
15. P. López, J.M. Drake, P. Pacheco, and J.L. Medina, An Ada 2005 Technology for Distributed and Real-Time Component-based Applications, in Proc. of the *13th Intl. Conference on Reliable Software Technologies Ada-Europe*, Venice, 2008