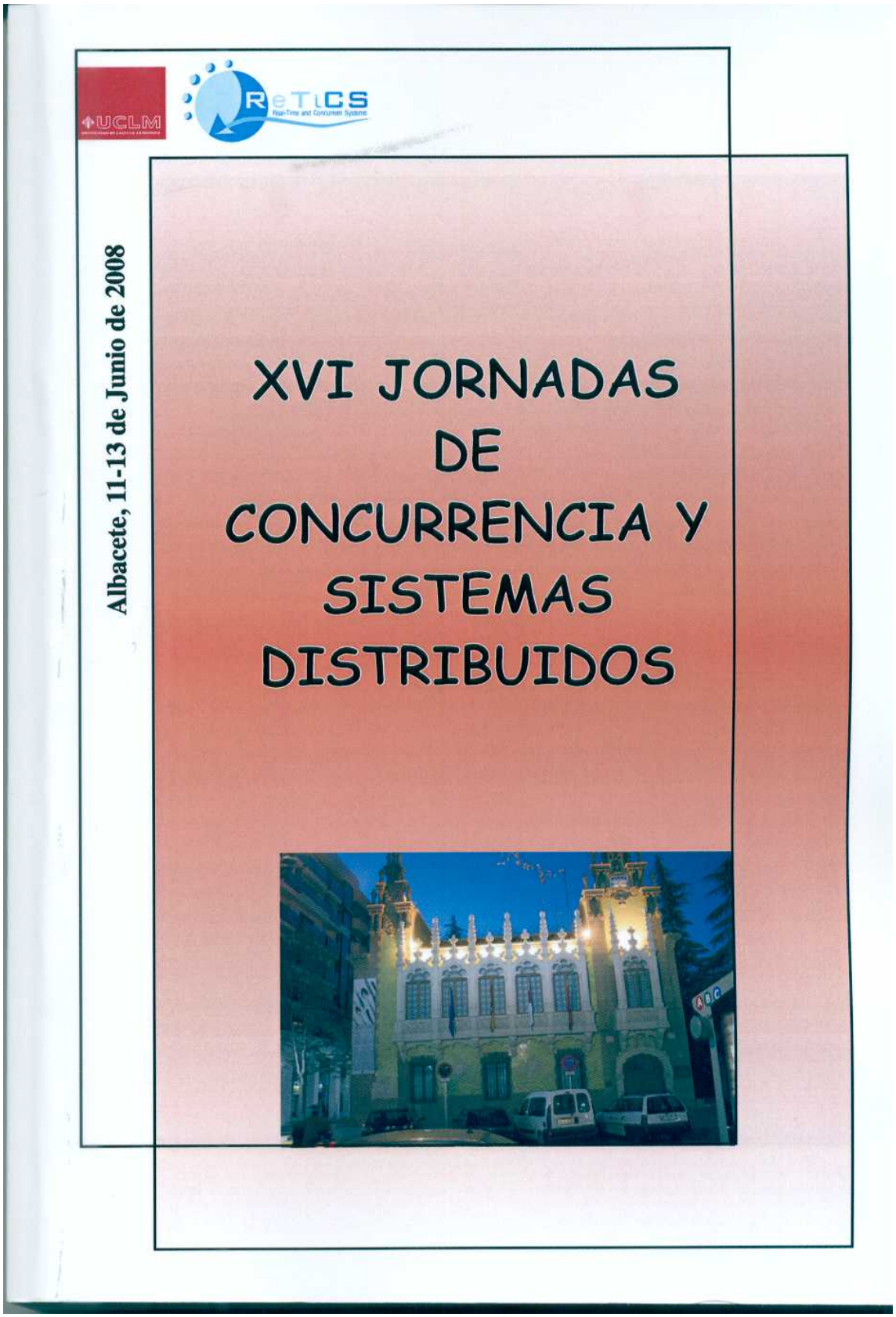

ISBN: 978-84-691-2813-8
Imprime: Imprenta GILSAN, Albacete
Impreso en España
Pgnas: 1-16

Tecnología de componentes CCM basada en conectores

Laura Barros, Patricia López, y José M. Drake

Avda. de los Castros s/n, 39005 Santander - Spain
{barrosl, lopezpa, drakej}@unican.es

Resumen. Se describe una tecnología para el desarrollo de aplicaciones distribuidas y de tiempo real basadas en componentes, que pueden ser ejecutadas en plataformas con diferentes soportes de comunicación. Hace uso de la especificación D&C de OMG para describir los componentes, las aplicaciones y las plataformas de ejecución. Utiliza el modelo de referencia ligero de CCM de OMG (LwCCM framework) como base de la arquitectura interna de los componentes y de las aplicaciones, pero elimina la dependencia del middleware CORBA sustituyéndolo por componentes distribuidos que se denominan conectores, cuyo código se genera de forma automática a partir de la especificación de los componentes y que encapsulan el mecanismo de comunicación que realiza la invocación remota entre componentes. La tecnología ha sido diseñada para garantizar requisitos de tiempo real en su especificación. La tecnología remarca e independiza las responsabilidades y el conocimiento que se requiere de los actores involucrados en el desarrollo de una aplicación basada en componentes. Para cada uno de ellos, define los productos de entrada y de salida sobre los que realiza su trabajo.

1. Introducción[1]

La creciente capacidad que ofrecen los procesadores y la gran cantidad de memoria de que disponen, hacen que cada día se les requiera mayor funcionalidad a los sistemas de tiempo real. Esto, junto a la naturaleza distribuida de las plataformas en que se ejecutan y los requisitos de tiempo real que deben satisfacer, hacen que el software de este tipo de sistemas sea cada vez más complejo. Para dar soporte a todos estos requisitos es necesario elaborar nuevas metodologías de desarrollo y nuevas herramientas que permitan abordar la creciente complejidad, simplificando los procesos de desarrollo y reduciendo los tiempos de respuesta a las demandas del mercado. Las tecnologías de componentes, que han sido exitosamente aplicadas en otros ámbitos de la indus-

[1] Este trabajo ha sido financiado por el Ministerio de Educación y Ciencia del Gobierno de España dentro del proyecto THREAD (TIC2005-08665-C03-02), por el Programa IST de la Comisión Europea dentro del proyecto FRESCOR (FP6/2005/IST/5-034026) y por la Red de Excelencia ARTIST2. Este trabajo refleja sólo el punto de vista de los autores; la UE no se responsabiliza del uso que se pueda hacer de la información contenida.

tria, aparecen como una de las estrategias con mayor futuro para dar respuesta a este reto.

El objetivo esencial de la metodología de componentes es poder construir aplicaciones ensamblando módulos software reutilizables, que han sido previamente desarrollados sin conocer las aplicaciones en las que se van a emplear. Los componentes deben haber sido diseñados para poder ser ejecutados en diferentes plataformas sin necesidad de modificar, y ni siquiera conocer, su código interno. Cuando se desarrolla esta estrategia en sistemas distribuidos, uno de los principales problemas es hacerlos compatibles con los diferentes sistemas de comunicación (middleware) que ofrece la plataforma. Toda tecnología de componentes está implementada sobre una plataforma que proporciona un middleware de base, a través del que se realizan todas las interacciones entre componentes, tanto locales como remotas. Sería deseable desarrollar una tecnología que, aún tomando un determinado middleware como base para las comunicaciones, permita la interacción entre componentes a través de otros mecanismos.

Las tecnologías que actualmente se dominan el mercado, como EJB o .NET, no son aplicables a sistemas de este tipo (distribuidos y de tiempo real), ya que por un lado no ofrecen prestaciones de tiempo real, y además, no son ni multiplataforma ni multilenguaje. La tecnología CORBA, sí permite la interacción entre componentes desarrollados en diferentes lenguajes y ejecutados en diferentes sistemas operativos, sin embargo, resulta complicado hacer predecible el comportamiento de una aplicación de tiempo real desarrollada con esta tecnología, y resulta demasiado pesada para aplicaciones que se ejecutan en entornos embebidos, en los que los recursos son limitados.

Varios proyectos europeos, MERCED[1], COMPARE[2] y FRESCOR[3] abordan, desde diferentes puntos de vista, el desarrollo de una tecnología de componentes compatible con los requisitos que presentan los sistemas embebidos, distribuidos y de tiempo real. Todos ellos toman como punto de partida el modelo CCM (Container/Component Model) que sigue la especificación LwCCM [4], pero eliminan la dependencia de CORBA como sistema de comunicación. Siguiendo este patrón, el código de negocio del componente se puede desarrollar de forma totalmente independiente de la tecnología subyacente, siendo el contenedor del componente el que ofrece todos los recursos necesarios para adaptar dicho código de negocio a la plataforma y tecnología correspondiente. Dicho contenedor es generado de forma totalmente automática a partir de la información introspectiva (metadata) que proporcionan los componentes.

La tecnología que se presenta en este artículo sigue también esta estrategia, añadiendo algunas características que la hacen especialmente compatible con sistemas distribuidos y de tiempo real:

- Como se ha dicho antes, no se utiliza CORBA como sistema de comunicación entre componentes. Cuando la conexión entre componentes requiera prestaciones proporcionadas por el middleware de base que se utiliza, se implementa con él. Sin embargo, cuando se requieren prestaciones especiales, la comunicación entre componentes se realiza a través de un tipo especial de componente, denominado conector, que engloba en su código los mecanismos de comunicación que se requieren. La conexión entre los componentes y el conector es siempre local.

- Se han añadido nuevos servicios y mecanismos en el contenedor que permiten controlar las características de planificación y los threads de los componentes, de manera que se consigue hacer predecible el comportamiento temporal de las aplicaciones.
- Se siguen los formatos definidos en la especificación D&C [5] de OMG para la descripción de las interfaces e implementaciones de los componentes, de las plataformas y de las aplicaciones. Esta especificación también ha sido extendida para que incorpore información introspectiva (metadata) acerca de características no funcionales de los componentes, en nuestro caso en concreto, información acerca del comportamiento temporal de los componentes y de las plataformas, que será utilizada para analizar la planificabilidad de la aplicación durante el proceso de desarrollo de la misma.

En este artículo nos vamos a centrar en el uso de conectores dentro de la tecnología y en el modelo estructural que resulta de su introducción. Con el empleo de conectores, se pueden abordar diferentes tipos de problemas que habitualmente surgen cuando se trata de adaptar una tecnología de componentes a sistemas distribuidos (sean o no de tiempo real):

- El middleware que se quiere emplear no está concebido para aplicaciones orientadas a componentes.
- Un componente requiere mecanismos de comunicación especiales, que no coinciden con los proporcionados por el sistema de comunicación de base.
- La tecnología se quiere ejecutar en entornos embebidos, que no son compatibles con middlewares complejos.
- Las aplicaciones a desarrollar tiene requisitos de tiempo real estricto, y requieren sistemas de comunicación que ofrezcan prestaciones de tiempo real.

Como prueba de la tecnología propuesta, dentro de nuestro grupo se han desarrollado dos implementaciones distintas:

- AdaCCM: es una implementación destinada a sistemas embebidos y con tiempo real estricto. No utiliza ningún middleware de base, sino directamente los recursos de concurrencia y distribución que ofrece el propio lenguaje Ada 2005 [6]. Las comunicaciones remotas se realizan a través de conectores desarrollados utilizando Ethernet estándar con el protocolo de tiempo real RTEP [7] y las aplicaciones se ejecutan sobre el sistema operativo de tiempo real MaRTE OS [8].
- hCCM: es una implementación con sólo requerimientos de tiempo real laxos que se ha construido utilizando el middleware ICE [9] de ZeroC. Admite componentes desarrollados en Java y C/C++ y plataformas de ejecución Linux y Windows.

El artículo se estructura del siguiente modo, en la sección 2 se explica el proceso de desarrollo de un componente como unidad que se puede distribuir y comercializar independientemente dentro de la tecnología. La sección 3 describe el modelo de referencia de la tecnología. La estructura interna de un componente se explica en la sección 4, mientras que en la sección 5 se describen las características esenciales de los conectores. La sección 6 muestra el proceso de generación de una aplicación como ensamblado de componentes previamente desarrollados, y finalmente, se exponen las conclusiones y líneas de trabajo futuro.

2. Proceso de desarrollo de un componente

La figura 1 describe las fases del proceso para desarrollar un componente distribuible, que puede ser empleado en futuras aplicaciones.

El proceso comienza cuando el especificador (*specifier)*, que es un experto del dominio de aplicación, elabora la especificación de un componente que satisface una funcionalidad requerida dentro del dominio. Dicha especificación se elabora de acuerdo a la especificación D&C, a través de un elemento del tipo Component Interface Description (fichero .ccd.xml). La descripción de la interfaz de un componente incluye los siguientes elementos:

- Los servicios que ofrece a otros componentes, agrupados en puertos denominados facetas. La funcionalidad de un puerto se define a través de la interfaz que implementa.
- Los servicios que requiere de otros componentes para implementar su funcionalidad, agrupados en puertos denominados receptáculos.
- Los atributos de configuración con los que se adapta cada instancia del componente a los requisitos específicos que se presentan en cada aplicación en la que se utilice.

Con el objetivo de diseñar componentes con comportamiento temporal predecible, se han ampliado los aspectos que se describen en la especificación del componente. Los componentes pueden ser activos, pero a fin de tener un control completo sobre los parámetros de planificación de sus threads, estos no se crean directamente en su código de negocio. Cuando un componente necesita un thread para desarrollar su funcionalidad, lo requiere del entorno de ejecución declarando un puerto de activación en su especificación. Estos puertos de activación implementan unas interfaces especiales definidas en la tecnología; el modo en que el contenedor los gestiona se explica en el apartado 4.

También se ha extendido la especificación D&C para incorporar en ella el modelo de comportamiento temporal del componente:

- Se describen los requisitos de componibilidad del componente si ha de mantener el comportamiento de tiempo real en la conexión.
- Se describen las transacciones que pueden iniciarse en él, y que serán utilizadas para definir la carga de trabajo de las aplicaciones.

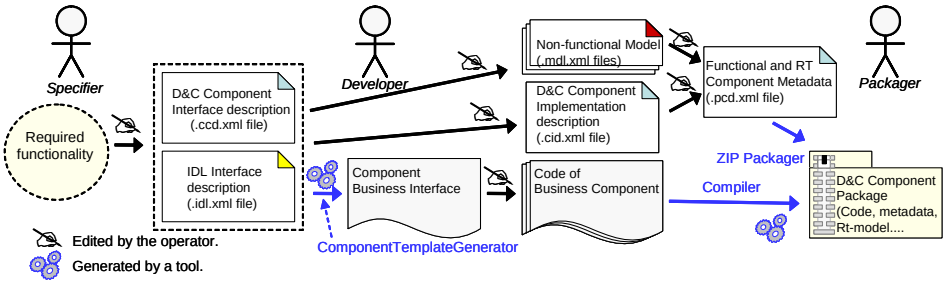

Fig. 1. Proceso de desarrollo de un componente

- Y también se describen los parámetros que adecuan el modelo de tiempo real al modo de operación de cada instancia.

En la figura 2 se muestran los elementos más relevantes de la descripción de un componente. Para hacer más sencilla la exposición utilizamos como ejemplo el componente SoundGenerator, cuya funcionalidad es generar sonidos bajo demanda de un cliente. Ofrece el puerto de negocio *playerPort* que implementa la interface *I_Player*. A través de él, los clientes requieren la generación de diferentes sonidos. Declara un receptáculo, *iLogger*, a través del que se conecta con un componente que ofrezca la interfaz *I_Logger*, que requiere para registrar los posibles errores que se puedan producir durante su operación. El componente es activo ya que requiere un thread propio para generar el sonido sin que el cliente tenga suspenderse a la espera de que el sonido haya sido generado. El thread se solicita declarando el puerto de activación *soundThread* que implementa la interfaz *PeriodicActivation* predefinida en la tecnología. El componente declara la propiedad del modelo de tiempo real *soundPlayerPeriod* que representa el periodo con el que el thread del entorno invoca el procedimiento *update()* del puerto de activacion *soundThread*.

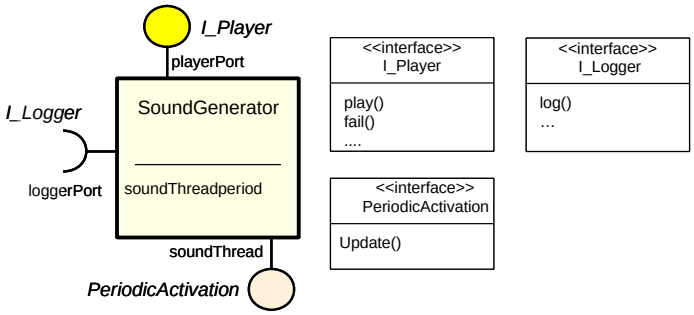

Fig. 2. Ejemplo de especificación de un componente

El desarrollador (*developer)* escribe el código de negocio del componente, como un conjunto de clases en el lenguaje de programación que elija, y como si fuera a ejecutarse siempre en entorno monoprocesador. El código de negocio deberá implementar una serie de métodos, que permitan al contenedor proporcionar los recursos de la plataforma que necesita y gestionar su ciclo de vida. Estos métodos se formalizan a través de la interfaz de gestión, denominada <*Nombre-component*>Managing, y cuyo código es generado de manera automática por la herramienta *ComponentTemplateGenerator*. Para ello, la herramienta toma como entrada la especificación del componente, junto con la descripción de las interfaces funcionales, formuladas en un lenguaje independiente del lenguaje de programación, por ejemplo IDL o Slice. El aspecto más importante de esta interfaz es que no presenta ninguna dependencia directa con la tecnología por lo que el desarrollador tiene libertad completa para desarrollar el código de negocio sin tener que conocer ningún detalle acerca del funcionamiento interno de la misma.

El desarrollador tiene un conocimiento detallado de la estructura y del código interno del componente, por lo que, junto con su código, debe elaborar los modelos que se requieren para su caracterización no funcional, por ejemplo, los modelos de tiempo real, calidad de servicio, etc. Además, debe especificar los recursos que el componen-

te requiere de la plataforma para poder ser ejecutado de forma correcta. Toda esta información se describe a través de un elemento del tipo Component Implementation Description (fichero .cid.xml), cuyo formato y contenido se define en la especificación D&C.

El empaquetador (*packager*) lleva a cabo la última fase del proceso, que consiste en construir el paquete con el que se distribuye y comercializa el componente como unidad que puede ser distribuida y comercializada de forma independiente. El empaquetador agrupa toda la información disponible del componente en un paquete que hace público para que pueda ser empleado en futuras aplicaciones. Este paquete contiene tanto el código del componente, como la información introspectiva (tanto funcional como no funcional) que permite al diseñador de una aplicación decidir la utilidad del componente en la misma. También se incluye la información que se requiere para poder instalar y ejecutar el componente sin necesidad de acceder a su código. Esta información se define de acuerdo al formato establecido en el elemento Package Configuration (fichero .pcd.xml) del D&C.

3. Modelo de referencia de la tecnología

La tecnología propuesta se basa en la reutilización del código de negocio de los componentes y en la generación automática del código de los contenedores que lo adaptan a la plataforma específica, y que incluyen los recursos necesarios para que los componentes se comuniquen entre sí. Para facilitar el proceso de generación de código, la tecnología se basa en un modelo de referencia bien definido, cuya estructura se muestra en la figura 3. Este modelo toma como punto de partida el modelo de contenedor/componente que se propone en la especificación LwCCM, pero se extiende con algunas características que favorecen la reutilización del código de negocio de los componentes, y el desarrollo de aplicaciones con comportamiento predecible.

Componente: Un componente es un módulo software reutilizable que ofrece una funcionalidad bien definida. Esta funcionalidad se describe a través del conjunto de servicios que ofrece a otros componentes, agrupados en puertos denominados facetas, así como, a través del conjunto de servicios que requiere de otros componentes para implementar su funcionalidad, agrupados en puertos denominados receptáculos. La funcionalidad de un puerto se define a través de la interfaz que implementa. Cada

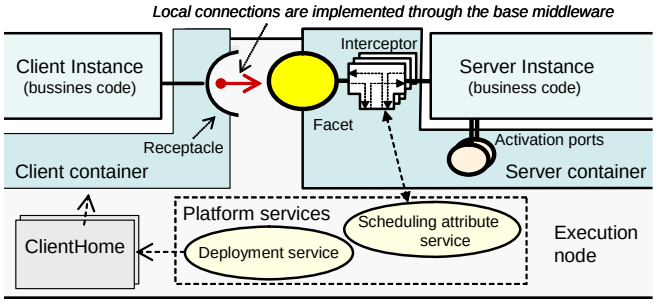

Fig. 3. Modelo de referencia de la tecnología

componente se divide en dos partes:

- El código de negocio: Es la parte del código que realmente implementa la funcionalidad ofrecida a través de las facetas del componente. Es desarrollado por un experto del dominio de la aplicación sin necesidad de conocer ningún detalle acerca de la tecnología subyacente, y de forma independiente al tipo de aplicación en la que va a ser utilizado.
- El contenedor: Su función es adaptar el código de negocio a la plataforma específica sobre la que ejecuta, y gestionar los mecanismos a través de los que el componente se conecta a otros componentes, de modo que el código de negocio permanezca inalterado. Su código es generado totalmente por medio de herramientas automáticas a partir de la especificación del componente.

Gestor del componente (Home): Cada tipo de componente tiene uno o más gestores, que constituyen los constructores (*factory*) de las instancias de los componentes de dicho tipo que se crean en un nodo de la plataforma. Son utilizados por las herramientas de despliegue para generar las instancias de componentes que forman parte de la aplicación.

Conectores: Es un elemento especializado que incorpora en su código el mecanismo de comunicación que se necesita para establecer la interacción entre un receptáculo de un componente cliente y una faceta de un componente servidor. La estructura del modelo de referencia cuando se conectan dos componentes a través de un conector se muestra en la figura 4. La funcionalidad que debe ser implementada por un conector es la serialización y deserialización de la información que intercambian, 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. Las características de los diferentes tipos de conectores que se pueden desarrollar se explican en el apartado 4.

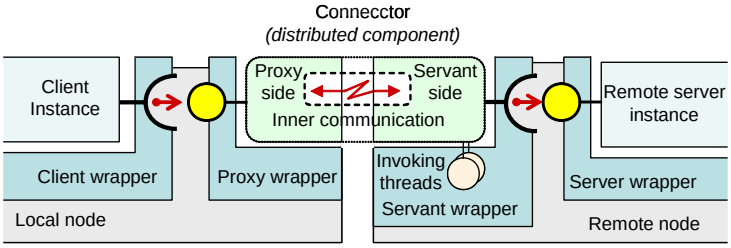

Fig. 4. Estructura del modelo de referencia con conectores

Interceptores: El concepto de intercepción se define en la especificación QoSforCCM [10]. Es el método utilizado para dar soporte a la gestión o control de características no funcionales de la aplicación. Permite introducir llamadas a servicios especiales del entorno en el proceso de invocación de una operación. La introducción de estos elementos es opcional y su configuración se realiza a través del plan de despliegue de la aplicación. Se introduce a nivel del contenedor, por lo que está oculto para el desarrollador del componente.

En nuestro caso, su función principal es controlar las características de planificación de los threads que ejecutan cada invocación. Basándose en los parámetros de

configuración que se le asignan en el plan de despliegue, cada interceptor conoce la prioridad con la que ha de ejecutarse cada invocación, por lo que hace uso del *SchedulingAttributeService*, para modificar los parámetros de planificación del thread que está ejecutando al valor correspondiente. Con esta estrategia, además de los mecanismos de asignación de prioridad tradicionales como ClientPropagated y ServerDeclared [11], se puede realizar una asignación de prioridades basada en el modelo transaccional de la aplicación [12], esto es, se ofrece la posibilidad de que una misma operación sea ejecutada con diferentes parámetros de planificación en función del punto, dentro de una misma transacción, desde la que es invocada.

SchedulingAttributeService: Es un servicio del entorno, especialmente diseñado para esta tecnología, que permite a los interceptores modificar los parámetros de planificación de los threads que ejecutan una invocación. El tipo de parámetro de planificación depende de la plataforma de ejecución, pudiendo ser prioridades, deadlines, contratos de servicio [13], etc.

5. Estructura interna de un componente

La arquitectura interna de la implementación de un componente se ha diseñado de modo que se incorpore el código de negocio sin necesidad de ser modificado, y sea más sencilla la generación automática del código de los elementos del contenedor, dejando al desarrollador del componente únicamente la responsabilidad del código de negocio. Dicho código puede ser elaborado, además, como si el entorno de ejecución fuese siempre monolenguaje y monoprocesador. Los elementos que forman la implementación de un componente se muestran en la figura 5.

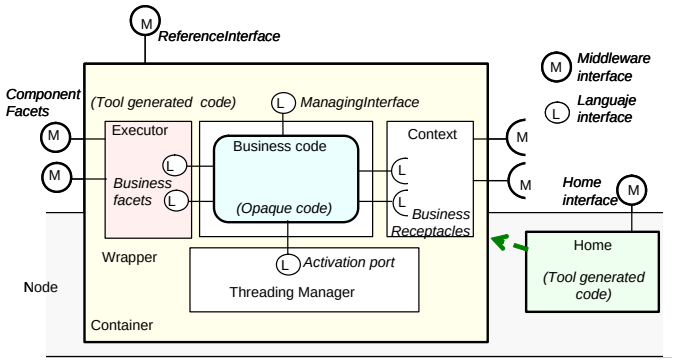

Fig. 5. Estructura de la implementación de un componente

El contenedor engloba todos los recursos que adaptan el código de negocio a la plataforma, siguiendo las reglas de interacción propuestas en LwCCM. Como se muestra en la figura 6 para el caso del componente SoundGenerator, está compuesto por varios elementos:

- El contenedor en sí (*Wrapper*), que ofrece la interfaz equivalente del componente. Dicha interfaz es la que LwCCM establece como la única interfaz a través de la que los clientes o la herramienta de despliegue acceden al componente. Esta inter-

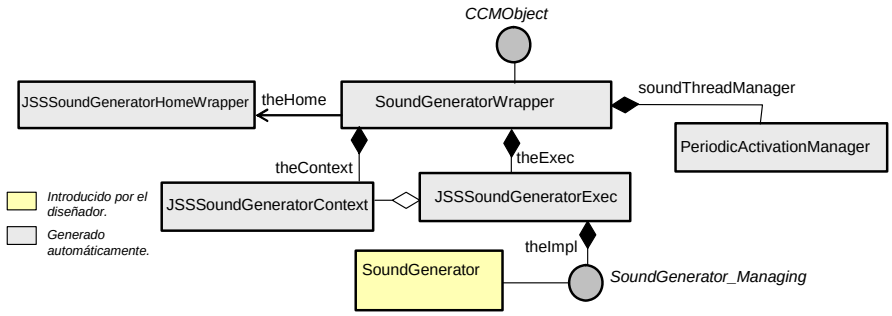

Fig. 6. Estructura del contenedor de un componente

faz está estandarizada, y extiende a la interfaz CCMObject, que entre otras, ofrece métodos para acceder a las facetas del componente, conectar componentes a sus receptáculos, etc. Su implementación dependerá del tipo de middleware de base de la tecnología (puede ser una interfaz en el caso de la implementación Ada, o un proxy en el caso de la implementación ICE).

- El ejecutor (*executor*), que implementa los servants o los adaptadores, a través de los que se invocan las operaciones de las facetas del código de negocio. La complejidad de este elemento depende del tipo de middleware empleado como base en la tecnología. En el caso de la implementación Ada, p.e., no se utiliza ningún middleware propiamente dicho, se trata simplemente de punteros Ada, por lo que esta clase es muy sencilla. La responsabilidad de la gestión de conexiones recae totalmente sobre los conectores. En el caso de la implementación ICE, por el contrario, es este elemento quien aporta todos los mecanismos ICE que se requieren para llevar a cabo la ejecución de las invocaciones, por lo que su complejidad aumenta.
- El contexto (*context*) es el mecanismo a través del que el código de negocio puede invocar operaciones en los componentes conectados a través de los receptáculos. Engloba los proxies que se requieren para traducir la invocación local del código de negocio, en una invocación (local o remota) basada en el middleware de base.
- El gestor de threads (*threading manager*) incluye todos aquellos elementos necesarios para gestionar los puertos de activación que se hayan declarado en el componente. Por cada puerto de activación, deberá crear un thread con sus correspondientes parámetros (el procedimiento a ejecutar, el parámetro de planificación correspondiente, el periodo en el caso de una activación periódica), y gestionarlo a lo largo de todo su ciclo de vida.

El código de negocio del componente sólo debe cumplir el requisito de implementar la interfaz de gestión (<*nombre componente*>.Managing), que es generada de forma automática y que no presenta dependencias con la tecnología subyacente. Esta interfaz se ha definido como parte de la tecnología y engloba todos los métodos que permiten al contenedor manejar el código de negocio durante todo el ciclo de vida del componente. Para la gestión del componente esta interfaz define los siguientes métodos:

- Métodos de navegación, *get*<*FacetName*>(), que permiten obtener cada una de las implementaciones de las facetas que ofrece el componente.

- Métodos de conexión, *set<ReceptacleName>*(), que permite conectar cada uno de los componentes servidores.
- Métodos de asignación, *set<AttributeName>*(), de los valores de los atributos de configuración del componente.
- Métodos de navegación, *get<ActivationPortName>*(), que permiten obtener las implementaciones de los puertos de activación definidos para el componente.
- Métodos para la gestión del ciclo de vida del componente.

A fin de diseñar componentes cuyo código tenga un comportamiento temporal predecible, no se permite crear threads en su código de negocio. En su lugar, cuando el componente necesita un thread para implementar su funcionalidad, declara un tipo especial de puerto, denominado puerto de activación, que implementa una de las dos interfaces *PeriodicActivation* o *OneShotActivation*. Cuando el componente es instanciado en una aplicación, el contenedor reconoce estos puertos, y por cada uno de ellos, le otorga al componente un thread, a través de la ejecución del correspondiente procedimiento implementado por el puerto con un thread creado por él. En el caso de un puerto de tipo PeriodicActivation, el thread invocará periódicamente el método *update()*, mientras que en el caso de un OneShotActivation, invocará una sola vez el procedimiento *run()*.

El desarrollador del código de negocio tiene total libertad para elaborarlo, sin embargo existen algunos aspectos de su estructura interna que deberían ser abordados en todos los casos:

- El componente deberá ofrecer implementaciones de cada una de las facetas ofertadas. En el caso de que se ofrezca una única faceta de una determinada interfaz, el propio componente puede implementarla directamente. Sin embargo, cuando un componente ofrece varias facetas con la misma interfaz, deberá incluirse un objeto independiente por cada una de ellas, como el objeto iPlayer_Impl en el caso del componente SoundGenerator, que se muestra en la figura 7. El acceso a estos objetos serán los que se retornen a través de los correspondientes métodos de navegación *getFacetName*.
- Lo mismo ocurre con los puertos de activación del componente, por cada uno de ellos, el código de negocio deberá instanciar un objeto que implemente la interfaz correspondiente, y en consecuencia, la funcionalidad asociada a dicho puerto.
- Las implementaciones tanto de facetas como de puertos de activación, operarán de acuerdo al estado del componente, que es único por cada instancia. Una estrategia adecuada de implementación consiste en agrupar el estado del componente en un objeto, que pueda ser accedido por todos los elementos, eliminando de este

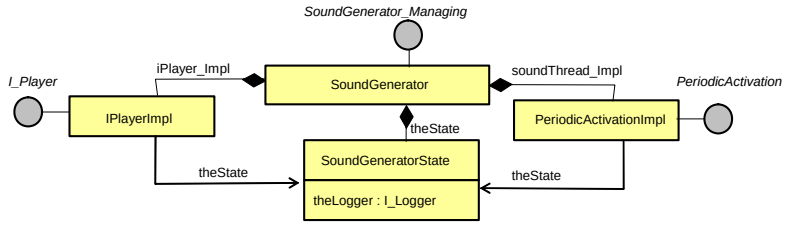

Fig. 7. Estructura del código de negocio de un componente

modo posibles dependencias cíclicas. Dentro del estado de un componente se almacenan todos los atributos de configuración definidos para el componente desde su especificación, los enlaces a los receptáculos, más toda la información que el desarrollador del componente considere necesaria para implementar la funcionalidad del componente (tanto de sus facetas como de los puertos de activación).

4. Funcionalidad y tipos de conectores

Cuando se utilizan los conectores como conexión entre dos componentes, la conexión entre el receptáculo del cliente y la faceta del componente Proxy del conector, y entre el receptáculo del componente servant del conector y la faceta del componente servidor, son ambas locales, e implementadas de acuerdo al middleware de base de la tecnología, como se muestra en la figura 3.

En esta tecnología un conector está compuesto por un componente (en el caso de conexión local) o por un par de ellos (en el caso de conexión remota). Las estructuras de estos son las mismas que la de cualquier componente, sólo que sus códigos son generados automáticamente por la herramienta de despliegue en función de la interfaz de los puertos que conecta y del medio de comunicación o middleware utilizado. Existen diferentes aspectos que deben ser abordados por un conector en función del tipo de conexión que realice:

- Con el objetivo de poder conectar componentes desarrollados en diferentes lenguajes de programación, el conector opcionalmente implementa los mecanismos de serialización y deserialización necesarios para que los argumentos de invocación y retorno puedan ser procesados por el correspondiente lenguaje.
- Debe implementar mecanismos de sincronización que permitan a los componentes clientes suspenderse a la espera de respuesta cuando realizan una invocación síncrona en el componente servidor.
- Debe ofrecer mecanismos para la ejecución remota de invocaciones, esto es, ofrecer threads que gestionen la ejecución de las invocaciones recibidas en el componente servant.
- Debe implementar el mecanismo a través del que las invocaciones son traducidas a mensajes que se envían a través de la red.

En función de la localización relativa de los componentes conectados, y el tipo de invocación de las operaciones (síncrona o asíncrona), aparecen diferentes tipos de conectores, como se muestra en la figura 8:

- En el caso de que la invocación sea local y síncrona, el conector, o no existe, o tiene una funcionalidad mínima. No son necesarios los mecanismos de sincronización, pues es el thread del cliente el que ejecuta la invocación, ni son necesarios los mecanismos de comunicación. Por el contrario, si los componentes están desarrollados en diferentes lenguajes de programación, el conector tiene que implementar los de serialización y deserialización de parámetros en la invocación y en el retorno.
- En el caso de que la invocación sea local pero asíncrona, el conector no requerirá mecanismos de comunicación, pero sí requerirá por cada invocación un thread

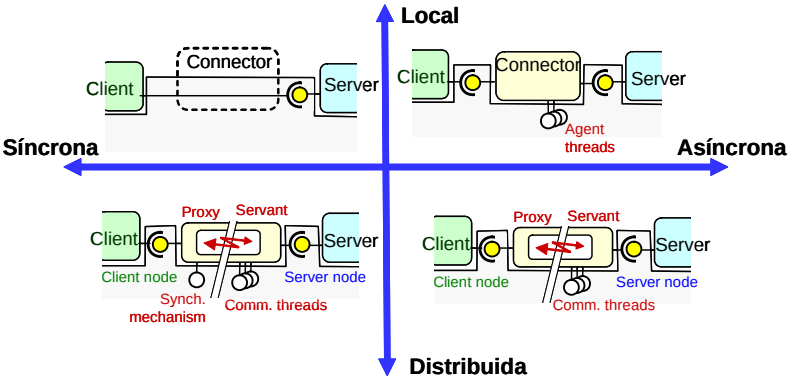

Fig. 8. Diferentes tipos de conectores

del entorno (a través de un puerto de activación) con el que llevar a cabo la operación invocada, y así, retornar de inmediato al cliente el control de flujo.

- Si el cliente y el servidor están instanciados en procesadores diferentes, como se muestra en la figura 3, el conector se compone de dos componentes: el componente proxy, que se instancia en el procesador del cliente, y el componente servant, que se instancia en el procesador del servidor. El proxy ofrece una faceta local (de la interfaz correspondiente) al componente cliente y contiene los mecanismos de sincronización para los threads que realicen invocaciones síncronas. El servant tiene un receptáculo que se conecta al componente servidor para invocarlo por delegación, y ofrece los threads que ejecutan las invocaciones recibidas. Los mecanismos de comunicación entre ambos componentes dependen del middleware escogido para la implementación.

5. Proceso de desarrollo de aplicaciones

El proceso de desarrollo de aplicaciones basadas en componentes, tal y como se muestra en la figura 9, incluye el proceso de diseño, configuración, despliegue y ejecución de una aplicación construida por ensamblado de componentes previamente desarrollados.

El ensamblador (*assembler*) describe la aplicación como un conjunto de instancias de componentes interconectadas entre sí, que implementan la funcionalidad requerida en su especificación. Los componentes son seleccionados de entre aquellos que se encuentran disponibles en el entorno de desarrollo. La descripción se realiza a través de un elemento del tipo Component Assembly Description (fichero .cad.xml), definido en el D&C. Para el caso de aplicaciones con requisitos de tipo no funcional (p.e. requerimientos temporales), esta descripción estructural debe ser complementada con la descripción de la carga de trabajo de la aplicación. La carga de trabajo corresponde al conjunto de transacciones que se ejecutan concurrentemente en ella [6], definiéndose para cada una de ellas los patrones de lanzamiento, el flujo de actividades a las que da lugar, y los requisitos que debe satisfacer. La especificación D&C ha sido extendida

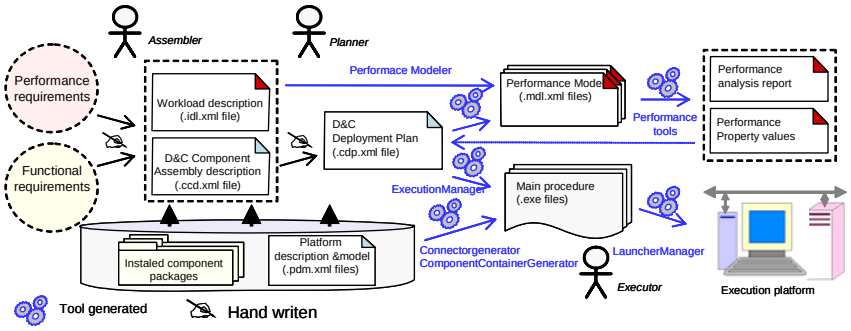

Fig. 9. Proceso de desarrollo de una aplicación basada en componentes

con el objetivo de poder describir cargas de trabajo asociadas a la descripción de una aplicación.

En la siguiente fase del proceso, el planificador (*planner*) parte de la descripción de la aplicación y construye un plan de despliegue para la misma. En él, se asigna cada instancia al nudo procesador en que se va a ejecutar, se formulan los valores de las propiedades de configuración de cada instancia de componente y se asigna el mecanismo de comunicación a utilizar en la interacción entre cada dos instancias. El resultado de esta fase es un elemento de tipo DeploymentPlan (fichero .pcd.xml) que describe completamente la aplicación, y el modo en que ésta va a ser ejecutada.

Si la aplicación tiene requisitos de tiempo real, en este momento debe construirse el modelo de comportamiento de la aplicación. Éste es generado por composición de los modelos de los elementos que forman parte de la aplicación, esto es, los componentes, la plataforma y los conectores. Sus correspondientes modelos de comportamiento deben estar almacenados en el entorno de desarrollo. Para cada carga de trabajo definida para la aplicación se puede elaborar un modelo de comportamiento que debe ser evaluado, y los resultados pueden ser utilizados para refinar los valores que se asignan a las propiedades de configuración de las instancias.

Finalmente el ejecutor (*executor*) hace uso de la herramienta de lanzamiento, que lleva a cabo las siguientes tareas:

- Una herramienta automática, *ComponentContainerGenerator*, genera a partir de la especificación del componente, el conjunto de ficheros de código a través de los que se implementa el contenedor del componente. Este contenedor aporta los mecanismos necesarios para adaptar el código de negocio previamente desarrollado, a la tecnología; su estructura se detalla en el apartado siguiente. Estos ficheros serán enlazados junto al código de negocio, para generar los correspondientes elementos que permitan ejecutar posteriormente el componente en cualquier aplicación.
- Genera el código de los conectores que se van a utilizar en la aplicación. La incorporación o no de un conector lo decide automáticamente la herramienta, en función de la ubicación de los componentes y de los atributos de la conexión. La herramienta ConnectorGenerator genera el código de todos los conectores de la aplicación, basándose únicamente en la información que aparece en el plan de despliegue:

  - Interfaz concreta de los puertos que conecta.
  - Tipo de sistema de comunicación definido para la conexión.
  - Parámetros de configuración, que serán específicos de cada tipo de middleware utilizado.
- Genera el código de los procedimientos principales a ejecutar en cada nodo para lanzar la aplicación. Estos procedimientos serán los encargados de instanciar, conectar y configurar los componentes y los conectores de acuerdo a la información que aparece en el plan de despliegue.
- El código de los procedimientos principales es compilado y enlazado con las librerías correspondientes a los componentes, y los conectores recién generados.
- Los ejecutables resultantes son transferidos y ejecutados en los correspondientes nodos.

6. Conclusiones

En este artículo se propone una tecnología de componentes especialmente concebida para el desarrollo de aplicaciones distribuidas y de tiempo real basadas en componentes. La tecnología parte del patrón contenedor/componente al que le añade ciertas características especiales que la adaptan a este tipo de sistemas. Por un lado, el empleo de conectores para la gestión de las comunicaciones independiza totalmente el código de negocio de los componentes del mecanismo de comunicación empleado, por lo que los componentes desarrollados pueden ser utilizados en plataformas que utilicen diferentes middlewares de comunicación. Por otro lado, se han añadido mecanismos y servicios al contenedor, que permiten hacer predecible el comportamiento de las aplicaciones, como es deseable en el caso de sistemas de tiempo real.

Se han desarrollado dos implementaciones de la tecnología propuesta con resultados satisfactorios. Una de las implementaciones con prestaciones de tiempo real, utilizando conectores implementados sobre un protocolo Ethernet de tiempo real y ejecutando sobre una plataforma mínima con un sistema operativo de tiempo real. Otra, sin prestaciones de tiempo real estricto, que adapta un middleware no orientado a componentes al desarrollo de aplicaciones basadas en este patrón. En este caso la tecnología utiliza ICE como middleware de base, y los componentes son desarrollados en lenguaje Java.

La especificación de los componentes, de las plataformas de ejecución y de las aplicaciones se realiza de acuerdo al estándar D&C de OMG. Se ha propuesto una extensión de tiempo real a la especificación, para que, en cada elemento se incorpore la información introspectiva que describe su comportamiento temporal, y que permite realizar análisis de planificabilidad durante el proceso de desarrollo de la aplicación.

Hay dos líneas de trabajo actualmente abiertas: la primera es el desarrollo de patrones de diseño de tiempo real compatibles con esta tecnología. La línea de trabajo consiste en el desarrollo de servicios a nivel del middleware que soporten el paradigma de reserva de recursos. Con ellos, será posible diseñar, planificar y configurar una aplicación de tiempo real sin tener en cuenta la carga de trabajo de la plataforma dis-

tribuida en que se ejecuta. La segunda va dirigida a la incorporación del paradigma MDA (Model Driven Architecture) a la tecnología. Con ella, se facilitará el desarrollo de las diferentes herramientas que se necesitan en los procesos de desarrollo de componentes y aplicaciones. Bastará definir los modelos de las estructuras de datos de las que se parte y a las que se llega, y una herramienta ayudará a generar las herramientas para cada middleware, lenguaje, sistema operativo o sistema de comunicación que se necesite.

References

[1] ITEA project MERCED (Market-Enabler for Retargetable COTS components in Embedded Domain). http://www.itea-merced.org.

[2] IST projects: "COMPARE (Component-based approach for real-time and embedded systems)". http://www.ist-compare.org.

[3] IST project: "FRESCOR (Framework for Real-time Embedded Systems based on Contracts)". http://www.frescor.org

[4] OMG: Lightweight Corba Component Model, ptc/03-11-03, November 2003

[5] OMG: Deployment and Configuration of Component-Based Distributed Applications Specification, version 4.0, Formal/06-04-02, April 2006

[6] T. Taft et al. editors: Ada 2005 Reference Manual. Int. Standard ISO/IEC 8652/1995(E) with Technical Corrigendum 1 and Amendment 1. LNCS 43-48, Springer-Verlag 2006.

[7] 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 Software Technologies, Ada-Europe 2005, York(UK), June 2005

[8] M. Aldea and M. González.: MaRTE OS: An Ada Kernel for Real-Time Embedded Applications. In: Proc. of the International Conference on Reliable Software Technologies, Ada-Europe 2001, Leuven, Belgium, Springer LNCS 2043, May 2001. http://marte.unican.es/

[9] M. Henning y M. Spruiell: "Distributed Programming with ICE". http://www.zeroc.com.

[10] OMG: "Quality of Service for CORBA Components", ptc/06-04-05. April 2006

[11] OMG: Real-Time CORBA Specification, v1.2 formal/05-01-04. Enero 2005

[12] M. González Harbour, J.J. Gutiérrez, J.C.Palencia and J.M.Drake: MAST: Modeling and Analysis Suite for Real-Time Applications. In: Proc. of the Euromicro Conference on Real-Time Systems, June 2001. http://mast.unican.es/

[13] Aldea M. et al: "FSF: A Real-Time Scheduling Architecture Framework" Proc. of 12th RTAS Conference, San Jose (USA), April 2006