RT-CCM: Tecnología de componentes de tiempo real basada en Ada 2005

Patricia López Martínez, Pablo Pacheco, Julio L. Medina y José M. Drake
Grupo de Computadores y Tiempo Real
Universidad de Cantabria
39005 Santander
lopezpa@unican.es, pablo.pacheco@gestion.unican.es, medinajl@unican.es, drakej@unican.es

Resumen

Se propone una tecnología para implementar aplicaciones de tiempo real basadas en componentes que utiliza el lenguaje Ada 2005, como medio de implementar los patrones de diseño de tiempo real, los mecanismos de comunicación entre componentes distribuidos y para hacerla compatible con sistemas embebidos que tienen procesadores con una dotación reducida de recursos. Los componentes se distribuyen con una especificación que sigue el estándar CCM del OMG, las aplicaciones se definen mediante un "plan de despliegue" que sigue el estándar de configuración y despliegue del OMG, y se despliegan mediante herramientas que a partir del plan de despliegue generan automáticamente el código Ada de la infraestructura, la cual proporciona el soporte de ejecución a los componentes y los mecanismos de comunicación entre ellos.

1. Introducción

Actualmente, a las aplicaciones de tiempo real se les requiere una funcionalidad que crece día a día, y asimismo, deben ejecutarse integradas en sistemas distribuidos complejos, y deben ejecutarse en procesadores embebidos en equipos e instrumentos con recursos hardware limitados. Para dar soporte a este perfil de requisitos se necesita desarrollar nuevas metodologías y herramientas capaces de gestionar su complejidad, simplificar los procesos de desarrollo y reducir los tiempos de respuesta a las demandas del mercado. La ingeniería de programación propone la metodología de componentes como un paradigma que da solución a estos requisitos. Sin embargo, las tecnologías que actualmente están en el mercado (EJB de Sun Microsystem, COM+ y .NET de Microsoft, o CORBA 3 de OMG) no son adecuadas para sistemas de tiempo real, y aún menos, si necesitan operar en una plataforma embebida.

Varios proyectos europeos MERCED[1], COMPARE[2] y FRESCOR[3] abordan desde diferentes puntos de vista el desarrollo de una tecnología de componentes para sistemas de tiempo real que sea compatible con plataformas embebidas. Utilizan el modelo denominado CCM (*Component/Container Model)* [4] en el que se utiliza la especificación CCM (CORBA Component Model) de OMG, pero sin requerir el uso de la tecnología CORBA [5] en su implementación.

En el modelo de un componente hay dos conceptos que son claves:

1. Contrato de uso: Los componentes son módulos software reutilizables y opacos, que ofrecen una funcionalidad de negocio que es configurable y está bien documentada. El diseñador construye las aplicaciones por composición de los componentes ya diseñados en base a la documentación que exhiben los propios componentes y sin necesidad de conocer su arquitectura ni su código interno.
2. Contrato de instanciación: Así mismo, los componentes incluyen en su descripción los recursos del entorno que necesitan para ejecutarse en una plataforma, así como de qué otros componentes necesitan hacer uso para elaborar su funcionalidad. Esta información es completa, y en base a ella, una herramienta puede generar de forma automática el código complementario que se necesita para que la plataforma dé soporte a su ejecución y para que los componentes interaccionen entre sí.

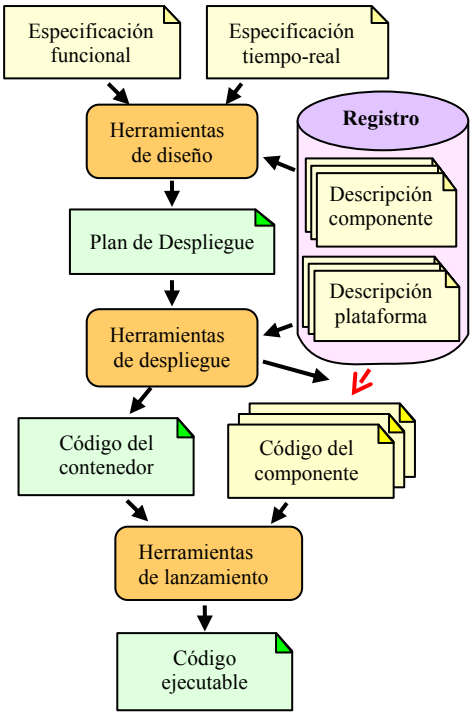

Figura 1. Proceso de desarrollo

En la figura 1 se muestra el proceso básico de diseño de una aplicación basada en componentes, que sigue el modelo CCM. El desarrollo de una aplicación se realiza en tres fases:

- En el entorno de diseño, el diseñador construye la aplicación en función de su especificación funcional, de los requisitos de tiempo real, de la descripción de los componentes de que dispone, y de la configuración de la plataforma en que se ha de ejecutar. El resultado del diseño es la elaboración del *Plan de Despliegue,* que describe la aplicación de acuerdo con la especificación D&C de OMG [6]. En ella se describen:
  - Las instancias de los componentes que constituyen la aplicación.
  - Los parámetros de configuración que se asignan a cada instancia.
  - La interconexión entre las instancias que se establecen.
  - La asignación a cada instancia de los recursos de procesamiento y comunicaciones que necesita.
- En el entorno de despliegue, la herramienta procesa el plan de despliegue de la aplicación:
  - Selecciona el código de los componentes que es adecuado a la plataforma en que se va a ejecutar.
  - Genera el código del entorno que da soporte a la ejecución de los componentes en cada procesador, esto es, que da acceso a los recursos de la plataforma, que asigna valores a los atributos de configuración y que proporciona los recursos de comunicación entre los componentes.
  - Elabora los requisitos que deben ser negociados con la plataforma para que la aplicación disponga de los recursos de procesamiento y de comunicación que necesita.
- En el entorno de ejecución, las herramientas transfieren el código en cada nodo de procesamiento, negocian con la plataforma la reserva de recursos de procesamiento y comunicación, instancian el código de los componentes y del entorno, y lanzan su ejecución.

Para aplicar este proceso al desarrollo de aplicaciones de tiempo real, se necesita extender las especificaciones CCM y D&C con nuevos elementos que describan el comportamiento temporal de los componentes y nuevas herramientas que evalúen los recursos que se requieren para satisfacer los requisitos temporales. Aspectos concretos de las especificaciones que se han introducido son:

- Los componentes incluyen como parte de su descripción un modelo que permite predecir su comportamiento temporal [7].
- La plataforma debe ser de tiempo real, y disponer de un modelo que describa sus recursos y el comportamiento temporal de sus servicios.
- En el diseño de las aplicaciones se tienen que especificar los parámetros de concurrencia, de planificación, de sincronización, etc. (*Threading Model*). Los valores que resulten de estos parámetros, deben estar incorporados en el *Plan de Despliegue* de la aplicación.

Cuando la plataforma de ejecución está constituida por computadores dotados con recursos suficientes, se puede utilizar las tecnologías CORBA de tiempo real [8], pero cuando se necesita ejecutar la aplicación en una

plataforma embebida, en la que los procesadores tienen los recursos limitados (especialmente si no dispone de sistemas de ficheros), la metodología CORBA es difícil de aplicar.

La reciente modificación de la especificación del lenguaje Ada, Ada 2005 [9], y el desarrollo de plataformas que lo soportan, introducen una nueva opción para implementar la tecnología de componentes CCM. Ésta es adecuada para plataformas embebidas, ya que basta que un procesador soporte el entorno de ejecución propio de Ada 2005 para que pueda ser utilizado con la metodología de componentes. El uso de Ada 2005 ofrece las siguientes ventajas:

- El lenguaje Ada ha sido concebido para el desarrollo de aplicaciones de tiempo real, y desde él, se puede formular el modelo de concurrencia, las políticas de planificación y los mecanismos de sincronización.
- El lenguaje Ada ofrece un modelo de distribución, y desde él, se pueden formular los mecanismos de interacción entre componentes desplegados en diferentes procesadores.
- En la versión Ada 2005 se introduce el concepto de interfaz, y a través de él, se pueden implementar la encapsulación de los servicios que ofrecen los componentes (*Facets* en CCM) o las referencias de los servicios que se requieren (*Receptacles* en CCM).
- En la versión Ada 2005 se introduce el concepto de tipos incompletos, lo que habilita la capacidad de especificar referencias cruzadas que son muy frecuentes en aplicaciones basadas en componentes.

En este artículo se propone una estrategia para implementar una tecnología de componentes CCM basada en componentes que se desarrollan utilizando el lenguaje Ada 2005, y que se distribuyen como paquetes Ada compilados. En esta tecnología, las herramientas de despliegue generan el código de los contenedores de la aplicación como código fuente Ada, y así mismo la comunicación entre los componentes se implementa utilizando las extensiones para sistemas distribuidos que define el lenguaje Ada.

El artículo se plantea en tres secciones. En el apartado 2 se plantea el modelo de tiempo real de los componentes CCM que se propone, haciendo énfasis en el modelo de concurrencia que se utiliza. En el apartado 3, se muestra a través de un ejemplo, el proceso y la información que se maneja en el desarrollo de una aplicación. El artículo concluye describiendo su estado actual de desarrollo y las expectativas que se tienen sobre los recursos que aún faltan.

2. Modelo de componente de tiempo real

La tecnología RT-CCM se basa en la reutilización (sin necesidad de modificación) del código de negocio de los componentes, y en la generación con herramientas automáticas del código que sirve de adaptación a la plataforma. A tal fin, se propone un modelo de referencia (*framework*) que trata de alcanzar los siguientes objetivos:

- Formular un modelo de los componentes que haga homogénea la adaptación de los componentes a la plataforma y la comunicación de los componentes entre sí.
- Definir para el entorno una estructura modular que simplifique la generación del código por las herramientas.

En la figura 2 se muestran los elementos básicos que constituyen el modelo de referencia de la tecnología RT-CCM. Estos son:

- Componente (Component): Son clases que extienden una clase raíz *CCM_Component*, heredando de ella un conjunto de puertos básicos (figura 3) que son utilizados por el entorno para su gestión y configuración. La característica propia de un tipo de componente es el conjunto de puertos de negocio que oferta (*Facets*). La funcionalidad de un puerto se define a través de la interfaz que implementa, y constituye una vista del componente que puede ser referenciada por otros componentes. Un componente puede requerir estar conectado a otros componentes para implementar su funcionalidad, estas referencias se denominan *Receptacles*.

  En RT-CCM un componente es un elemento pasivo que sólo ofrece operaciones a través de sus puertos. En su cuerpo puede disponer de elementos de sincronización que regulen la interacción entre los *threads* que invocan las operaciones de una instancia del componente.
  La actividad de un componente se declara mediante las *interfaces de activación*. Estas interfaces son reconocidas por el entorno, y a través de ellas, le proporciona los *threads* que requiere para su operación. Se definen dos tipos de interfaces de activación: la denominada

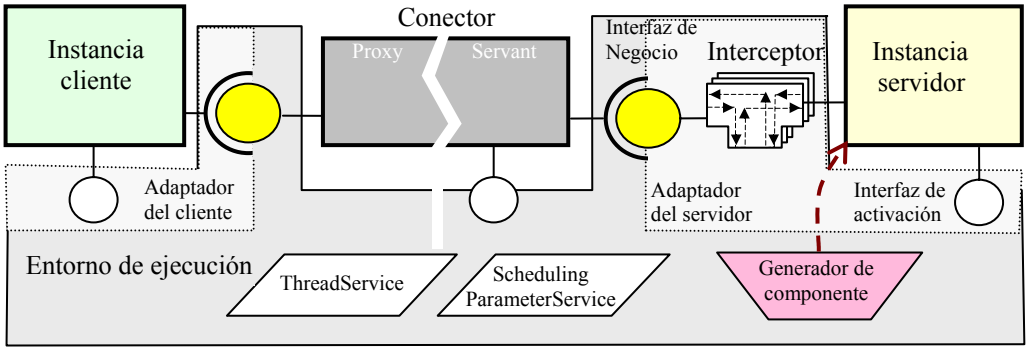

Figura 2. Modelo de referencia de la tecnología RT-CCM.

OneShotActivation que declara un procedimiento *run()* que es invocado por el entorno en el proceso de lanzamiento de la aplicación, y la interfaz denominada *Periodic Activation* que declara un procedimiento *update()*, que el entorno invocará periódicamente, con un periodo definido en configuración. Un componente puede implementar múltiples puertos de activación y las invocaciones que recibe por cada una de

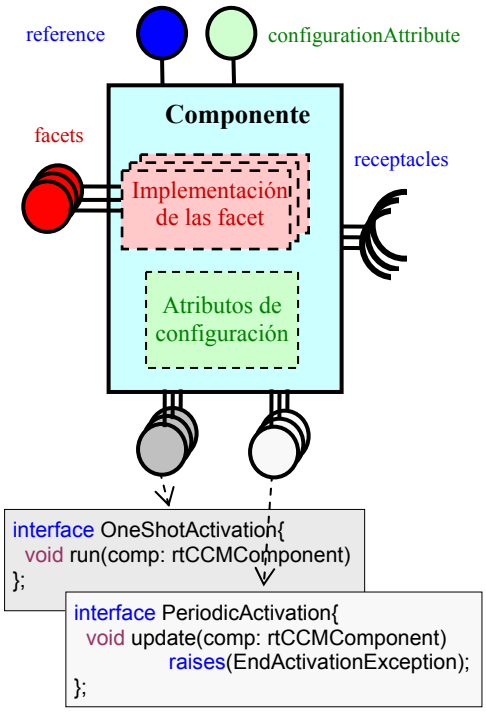

Figura 3. Interfaces de los componentes.

ellas constituyen un nivel independiente de concurrencia que el componente gestiona.

- Generador de Componente (*Home*): Es un modulo asociado a un tipo de componente, y constituye el generador (*factory*) de las instancias de ese tipo en cada procesador.
- Adaptador de componente (*Wrapper*): Es el código que adapta cada instancia de componente a la plataforma en que se ejecuta. Proporciona la interfaz a través de la que la instancia del componente accede a los componentes que se conectan a sus *receptacles*.
- Conector (Connector): Constituye el elemento a través del que un componente se comunica con otro componente con el que está conectado a través de un puerto. En la tecnología RT-CCM un conector es un componente más, sólo que su código es generado automáticamente por la herramienta de despliegue en función de la interfaz de los puertos que conecta, de la localización relativa de ambos componentes, y del medio de comunicación que se utilice.

En la figura 4 se muestran algunas opciones de implementación de un conector. En el caso de que la invocación sea local y síncrona (a), el conector no existe, y el cliente invoca directamente el puerto del servidor. En el caso de que la invocación sea local pero asíncrona (b) el conector requiere del entorno un *thread* con el que se lleva a cabo la operación invocada, y retorna de inmediato al cliente el control de flujo. Si la invocación es distribuida (c) por estar el cliente y servidor instanciados en procesadores diferentes, el conector se compone de dos elementos: el elemento *proxy* se instancia en el procesador del cliente y el elemento *servant* se instancia en el procesador

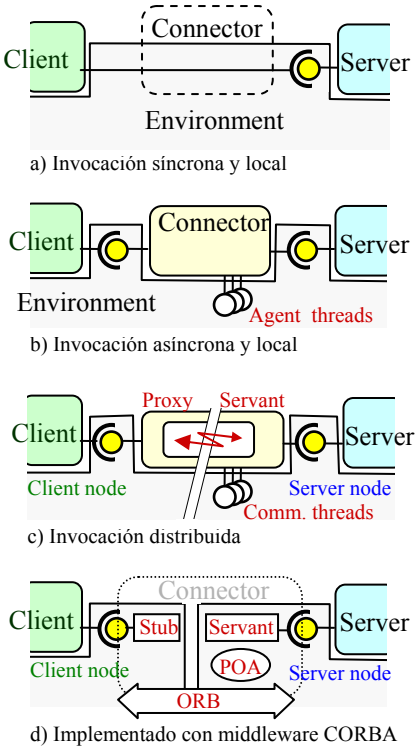

Figura 4. Opciones del conector.

del servidor. Entre ellos se establece la comunicación por el medio que se haya especificado. En este caso, el conector puede implementar un mecanismo de invocación síncrono, suspendiéndose el cliente en el Proxy hasta que la invocación finaliza, o implementar un mecanismo asíncrono y el cliente continúa su ejecución mientras que la operación invocada se ejecuta en un *thread* proporcionado por el entorno. La tecnología RT-CCM también puede hacer uso del middleware de CORBA para implementar el conector (d).

- Interceptor (Interceptor): Es un elemento que se introduce individualmente por cada operación que lo requiera. En la tecnología RT-CCM se utiliza, entre otras funciones, para controlar las características de planificación del *thread* que ejecuta la operación que se invoca. A través de ellos existen tres políticas de asignación de los parámetros de planificación:
  - Parámetros transmitidos por el cliente: Los parámetros de planificación con los que se invoca una operación son los del cliente que invoca (o los equivalentes en el procesador remoto si la invocación es distribuida).
  - Parámetros establecidos en el servidor: Los parámetros de planificación están definidos en el servidor y son los mismos en cualquier invocación.
  - Parámetros definidos por la transacción. Los parámetros de planificación de cada invocación se establecen en función de la transacción y de la actividad a las que corresponde la invocación. Este modo hace posible planificar de forma detallada y eficiente las aplicaciones de tiempo real.

En la figura 5, se muestran los elementos básicos que constituyen un interceptor. Se introducen las operaciones *Receive_Request()* y *Send_Reply()* en las secuencias de invocación y respuesta de la operación. Estas operaciones se introducen a nivel del *wrapper*, y por tanto, desde ellas se tiene acceso a los recursos del entorno, y en particular al servicio *SchedulingParameterService* con capacidad de modificar los parámetros de planificación del *thread* que lo invoca, y que en este caso es el *thread* que invoca la operación del servidor.

La gestión de los parámetros de planificación se realiza utilizando el identificador *StimulusID,* que es transmitido por la invocación e identifica al cliente que lo invoca y la transacción y actividad del modelo reactivo a la que corresponde:

- En la operación *Receive_Request()* el interceptor hace uso de la tabla que le ha

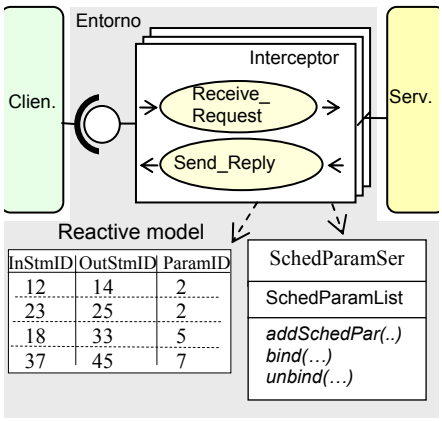

Figura 5. Elementos de un interceptor.

sido asignada en configuración, y en función del *stimulusID* asociado a la invocación, se obtiene el nuevo *stimulusID* que será asociado a la actividad que se inicia, y el identificador del parámetro de planificación *(schedParamID*) que define las características de planificación que se asocian al *thread* que ejecuta la operación. Haciendo uso del servicio *Scheduling ParameterService*, el *thread* adquiere las características de planificación que corresponden al *schedParamID* que le ha sido asignado.

- En la operación *send_Reply()* el *thread* recupera el *stimulusID* y los parámetros de planificación que tenía en la invocación.

- *ThreadService*: Es el servicio que gestiona los *threads* que ejecutan las actividades de la aplicación. Los *threads* son aplicados por el entorno a los componentes (dentro de los que se incluyen los conectores) de acuerdo con los puertos de activación que tengan definidos.
  Como se muestra en la figura 6, cada *thread* contiene una tarea Ada, que permanece suspendida en espera que mediante el procedimiento sincronizado *Start()* se establezca la operación que debe ejecutar, los parámetros de planificación con que debe operar, y su carácter de ejecución periódica o

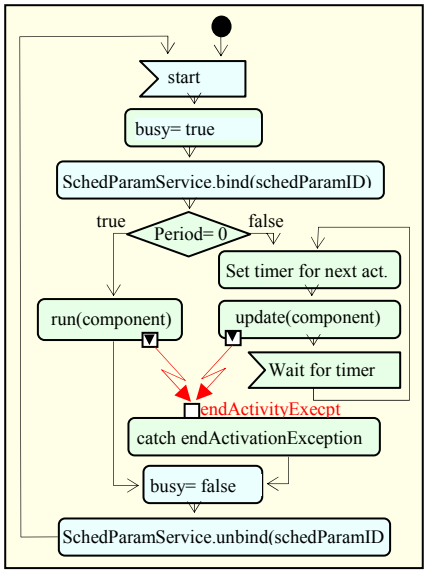

Figura 6. Actividad de un *thread*.

única. Una tarea finaliza cuando desde dentro del componente se eleva la excepción *EndActivationException*.

3. Ejemplo de aplicación: Track follower

Con el fin de mostrar una aplicación típica de la tecnología RT-CCM, así como el procedimiento que se sigue en el desarrollo de una aplicación basada en ella, se describe en esta sección un ejemplo que hace uso de muchas de sus capacidades.

La aplicación tiene como función controlar la posición de un robot a fin de que siga las posiciones que esporádicamente le proporciona un dispositivo de localización de objetivos (Tracker).

En la figura 7 se muestra la arquitectura de la aplicación. Se compone de cinco componentes, cuatro de ellos son reutilizados y están definidos con independencia de la aplicación, y el quinto es el cliente, que como es habitual, es específico de la aplicación. Para comprender mejor la operación de los componentes y de la aplicación, también se muestran en las figuras los elementos que ofrecen las interfaces. Los componentes que se utilizan son:

- TrackFollower: Es el cliente específico de la aplicación, y representa un módulo ligero basado en una Interfaz Grafica (GUI) que tiene capacidad para ejecutar concurrentemente las siguientes actividades:
  - Atender el teclado y en función del comando que se pulsa ordenar su ejecución.
  - Controlar al dispositivo Tracker y cuando éste propone un objetivo, generar la trayectoria que debe seguir el robot para alcanzarlo.
  - Quedar a la espera de las alarmas que se registran en el *logger*, y cuando se producen monitorizarlas en pantalla.

  Es un objeto activo que requiere tres *threads* del entorno por los puertos: *keyboardThread* (actividad de la GUI, 200ms), *alarmMonitor* (que se suspende en el *logger* para monitorizar las alarmas que se produzcan) y *trackerThread* (que gestiona el dispositivo *tracker* y responde a sus consignas).
- ServosController: Es un componente de tiempo real que ejecuta periódicamente el algoritmo PID de control de n servos. Tiene capacidad de

registrar bajo demanda la trayectoria de los servos en un *logger*.

Es un objeto activo, ya que es capaz de mantener de forma autónoma el control de los servos y el registro de su trayectoria en el *logger*. A tal fin, requiere del entorno dos activaciones periódicas representadas por los puertos *controlThread* (5 ms) y *loggerThread* (1000 ms).

- SoundGenerator: Es un componente capaz de generar sonidos y melodías utilizando el altavoz interno y el temporizador del PC.

  Es un componente activo que se basa en una tarea periódica que modifica esporádicamente la frecuencia del temporizador que determina el sonido que se genera. A tal fin, tiene definido el puerto *soundThread* (125 ms).
- DataBase: Es un componente que se basa en una base de datos comercial. Ofrece una vista de *logger* a través de la interfaz *I_Logger*, que permite registrar con marcas de tiempo tres tipos de eventos: alarmas, eventos y avisos. Es un objeto pasivo.
- PCI9111Card: Es un objeto pasivo que permite el acceso a la tarjeta de entrada/salida comercial PCI9111. La interfaz I_AnalogIO permite leer y establecer líneas analógicas de entrada y salida.

El modelo de tiempo real de esta aplicación resulta de identificar las transacciones que concurren en su operación. Las transacciones tienen dos orígenes. Unas se generan en los componentes clientes y resultan de eventos externos (p.e. requerimientos del *tracker*) o de situaciones que se generan como consecuencia del procesamiento de datos. Otras se generan en los componentes clientes o servidores (como consecuencia de activaciones que se ejecutan periódicamente con origen en activaciones periódicas realizadas desde el entorno). En la aplicación *TrackFollower* se pueden identificar las siguientes transacciones:

- Comando del operador: Tiene su origen en el componente *TrackFollower*, tiene naturaleza periódica y se produce en cada activación *keyboardThread* (200 ms). Su plazo de conclusión es el propio periodo.
- Procesamiento de objetivos del *tracker*: Tiene su origen en el componente *TrackFollower*. Se generan esporádicamente cada vez que el *tracker* determina un nuevo objetivo a seguir (>2000 ms). Su plazo de conclusión es función

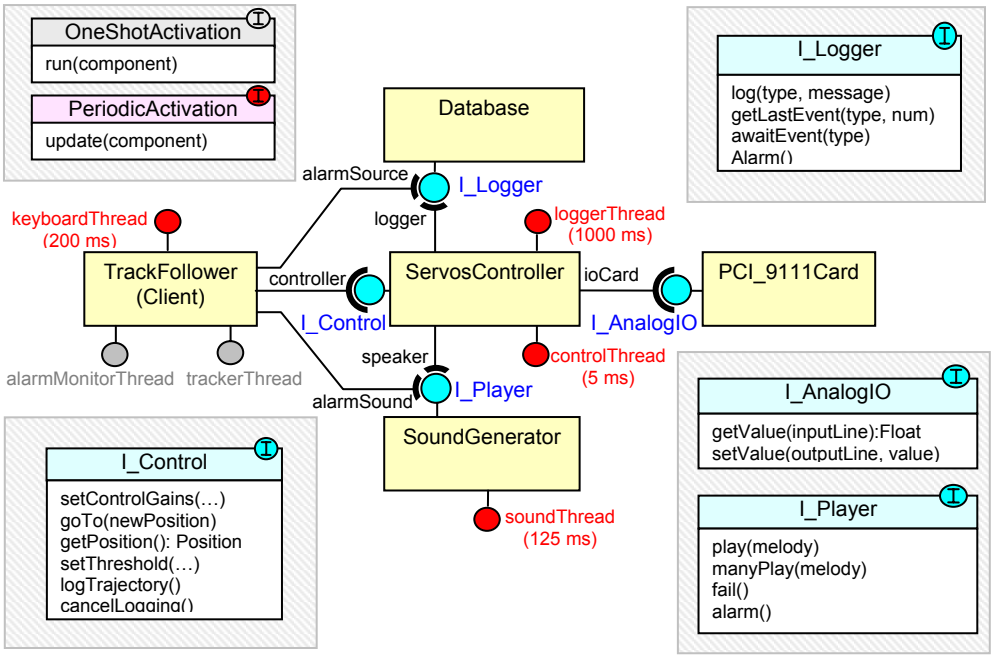

Figura 7. Arquitectura de la aplicación TrackFollower.

del tiempo de respuesta que se requiere para alcanzar el objetivo (<500 ms).

- Control de servos: Tiene su origen en el componente *ServosController*. Se genera periódicamente con la activación *controlThread* (5 ms). Su plazo de conclusión es el propio periodo.
- Registro de la trayectoria: Tiene su origen en el componente *ServosController*. Se genera periódicamente con la activación *loggerThread* (1000 ms). Su plazo de conclusión es el periodo.
- Generación de sonido: Tiene su origen en el componente *SoundGenerator*. Se genera periódicamente con la activación *soundThread* (150 ms). Su plazo de conclusión es el periodo.

4. Estado de ejecución y conclusiones

La tecnología que se propone representa una opción útil para el desarrollo de aplicaciones de tiempo real en sistemas embebidos, en los que a menudo el hardware está limitado en recursos y no es estándar. Al estar el lenguaje Ada específicamente adaptado para desarrollar aplicaciones en este tipo de plataforma, y soportar directamente desde el lenguaje los patrones de tiempo real y los mecanismos de distribución, la tecnología RT-CCM resulta ser muy ligera.

Las pruebas se han realizado utilizando la arquitectura de MaRTE OS operando como una aplicación Linux, y utilizando el compilador GNAT (GAP). Esta opción es válida para el desarrollo de la tecnología, ya que desde el punto de vista de la arquitectura software es totalmente equivalente. Sin embargo, no es una plataforma de ejecución adecuada para su validación temporal, ya que al ejecutarse la aplicación como un proceso estándar Linux, el entorno de ejecución no es de tiempo real estricto.

La tecnología que se propone necesita de un conjunto de recursos entre los que algunos aún no están disponibles:

- Un sistema operativo de tiempo real que soporte la ejecución de aplicaciones Ada 2005 y sus extensiones de tiempo real.
- Un sistema de comunicaciones de tiempo real que soporte las extensiones de Ada Distribuido y que sea compatible Ada 2005.

Actualmente no se dispone de ambos recursos, aunque dentro del proyecto THREAD se espera en pocos meses disponer de ellos para la plataforma MaRTE OS.

Agradecimientos

Este trabajo ha sido realizado en el marco del proyecto THREAD (TIN 2005-08665-C03-02) financiado por la Comisión Interministerial de Ciencia y Tecnología.

Referencias

[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] Ansgar Radermacher, Sylvain Robert, Sébastien Gérard, François Terrier, Vincent Seignole, Virgine Watine. : “Separation of Concerns Due to Extensible Container Support” Proceedings of 4th International Workshop on Adaptative and Reflective Middleware, December 2005.

[5] OMG: Object Management Group, “Common Object Request Broker Architecture,” version 3.0.3, OMG document number formal/04-03-01.

[6] OMG : "CORBA Components Model Specification", version 1.2, Formal/05-01-04.

[7] P. López, J.M. Drake, and J.L. Medina: “Real-Time Modelling of Distributed Component-Based Applications” Proc. of 32h Euromicro Conference on Software Engineering and Advanced Applications, Croacia, August 2006.

[8] OMG: “Real-time CORBA specification” Version 4.0. Formal/06-04-02.

[9] S. T. Taft, R. A. Duff, R. L. Brukardt, E. Ploedereder, and P. Leroy, editors. Ada 2005 Reference Manual. International Standard ISO/IEC 8652/1995(E) with Technical Corrigendum 1 and Amendment 1. Number 4348 in Lecture Notes in Computer Science. Springer-Verlag, 2006.