RT-GLADE[1]: Implementación Optimizada para Tiempo Real del Anexo de Sistemas Distribuidos de Ada 95

Juan López Campos, J. Javier Gutiérrez, y Michael González Harbour

Departamento de Electrónica y Computadores
Universidad de Cantabria
39005 - Santander, SPAIN
{lopezju,gutierjj,mgh}@unican.es

Resumen[2]. Este trabajo presenta una modificación de GLADE —la implementación actual de GNAT del anexo de sistemas distribuidos (DSA) de Ada 95 (Anexo E)— para soportar el desarrollo de aplicaciones distribuidas con requisitos de tiempo real. La modificación de esta implementación, que llamamos RT-GLADE (Real-Time GLADE), está especialmente indicada para aplicaciones empotradas compuestas por un conjunto pequeño y heterogéneo de procesadores y redes de comunicaciones, y asegura un comportamiento temporal predecible. Un modelo de tiempo real de la implementación permite al programador de la aplicación determinar y optimizar el comportamiento temporal de todo el conjunto de la aplicación, mediante la aplicación de las correspondientes técnicas de análisis de planificabilidad y de asignación de prioridades. Esta versión de tiempo real de GLADE continúa siendo conforme al DSA, por lo que la aplicación completa se puede desarrollar en el contexto del lenguaje Ada 95.

Palabras clave: tiempo real, sistemas empotrados, sistemas distribuidos, Ada 95, modelado y análisis de planificabilidad.

1 Introducción

En los últimos 20 años, el concepto y la necesidad de distribución en los sistemas computacionales apenas ha cambiado, pero sí que lo ha hecho la tecnología que nos permite realizar esta distribución, que brinda ahora un amplio abanico de nuevas posibilidades. Así, hemos podido ver cómo los paradigmas de distribución han ido evolucionando desde los mecanismos de paso de mensajes a las llamadas a procedimientos remotos (RPCs), a la distribución de objetos, o últimamente a la distribución de componentes. Esta necesidad de distribución la recogió el estándar Ada 95 [16] en su Anexo E, y sin duda fue fruto de la necesidad que tenían los usuarios de Ada de implementar este tipo de sistemas. Por ejemplo, hasta la aparición del estándar

1. Este artículo es una versión preliminar con nuevos resultados del trabajo presentado a 9th International Conference on Reliable Software Technologies, Ada-Europe 2004 (Palma de Mallorca, 14-18 de junio de 2004.

2. Este trabajo ha sido financiado en parte por la *Comisión Interministerial de Ciencia y Tecnología* (CICYT) en el proyecto de investigación con referencia TIC2002-04123-C03-02 (TRECOM), y por el programa IST de la Comisión Europea en el proyecto de investigación con referencia IST-2001-34820 (FIRST).

Ada 95, sólo en la revista ACM SIGADA Ada-Letters aparecieron al menos 38 artículos relativos a sistemas distribuidos y Ada en diferentes aspectos como tiempo real, tolerancia a fallos, comunicaciones, o modelado.

Sin embargo, la realidad es que este anexo ha tenido una repercusión mínima en los diferentes sectores que se dedican al desarrollo de aplicaciones distribuidas. Se nos pueden ocurrir múltiples causas, pero quizá la más importante pueda ser que frente a la implementación del Anexo E, incluso aquellos que trabajan con Ada siempre han podido encontrar una alternativa que se adaptaba mejor al nicho concreto de la aplicación que querían desarrollar. Por ejemplo, para la distribución sin más, podríamos optar incluso por cambiar de lenguaje de programación utilizando Java, que aporta una infraestructura de distribución de objetos más moderna. Si queremos integración con software desarrollado en otros lenguajes y para diferentes plataformas podríamos optar por la vía de distribución de objetos de CORBA [10]. En este último punto también hay trabajos en los que se han discutido y planteado estrategias de integración entre los mundos de Ada y CORBA [12][15].

Quizás la razón más importante por la que el DSA no se ha utilizado mucho sea porque en el entorno de aplicaciones en el que Ada tiene gran implantación, los sistemas de tiempo real, el DSA no proporciona suficiente grado de control y predecibilidad temporal. Creemos que si el DSA soportara la distribución de aplicaciones de tiempo real, podría utilizarse en aquellos entornos en los que se elige Ada por ser el mejor lenguaje de programación para sistemas de tiempo real.

El objetivo de este artículo es obtener una implementación del actual Anexo E que pueda ser utilizada por las aplicaciones de tiempo real, demostrando las modificaciones al DSA propuestas en [6] son viables. Las propuestas de ese artículo están destinadas al entorno de aplicaciones en el que solemos trabajar: sistemas de control industrial empotrados (robots o sistemas de inspección visual) compuestos por varios procesadores distribuidos conectados por una o más redes de comunicación, en los que es necesario garantizar requisitos temporales.

Hay varias implementaciones del DSA que soportan particionamiento y distribución de aplicaciones Ada. Una de estas implementaciones es GLADE, desarrollado en sus inicios por Pautet y Tardieu [11][13], y que se incluye actualmente en el proyecto GNAT desarrollado por Ada Core Technologies (ACT) [1].

La implementación que presentamos en este artículo se llama RT-GLADE (Real-Time GLADE), porque ha sido construida modificando GLADE para mejorar sus características de tiempo real. Las modificaciones realizadas están basadas en trabajos que hemos realizado anteriormente en los que se estudiaba la convergencia hacia un anexo distribuido de tiempo real para Ada [5][6]. Además, para poder desarrollar aplicaciones de tiempo real también es necesario que RT-GLADE disponga de redes que soporten comunicaciones en tiempo real. En este trabajo se integra en RT-GLADE la red de tiempo real RT-EP [8], que implementa un protocolo de paso de testigo sobre hardware Ethernet estándar. Otro aspecto importante en este tipo de sistemas es la utilización de un sistema operativo de tiempo real. Hemos implementado RT-GLADE sobre MaRTE OS [2], que es un sistema operativo de tiempo real que cumple con el

Perfil Mínimo de POSIX [14]. Como MaRTE OS se basa en la versión 3.15p del compilador de GNAT, hemos modificado la versión 3.15p de GLADE, la última versión publica disponible que es compatible con ese compilador.

El haber construido todas las partes del sistema con comportamiento de tiempo real, hace posible que se puedan utilizar las herramientas y estrategias de modelado, análisis de planificabilidad, y optimización de la asignación de prioridades, que permiten una correcta caracterización de la respuesta temporal [4][9][7].

El artículo está organizado de la siguiente manera. En la sección 2, mostramos una visión general de la arquitectura y las principales características de GLADE relacionadas con el comportamiento de tiempo real y su cumplimiento del DSA. La sección 3 muestra las nuevas características introducidas en la implementación de RT-GLADE, así como el subconjunto de parámetros de configuración que pueden utilizarse para aplicaciones de tiempo real. También muestra algunos detalles del subsistema de comunicaciones que soporta requisitos de tiempo real. En la sección 4 se presenta un ejemplo sencillo en el que se muestra cómo utilizar RT-GLADE para construir la aplicación. La sección 5 recoge algunas medidas que ayudan a evaluar los beneficios de nuestra implementación. Por último, en la Sección 6 se exponen nuestras conclusiones.

2 La distribución en Ada y GLADE

Para Ada 95 [16], un sistema distribuido se define como la interconexión entre uno o más nodos procesadores, y cierta cantidad, cero o más, de nodos de almacenamiento, que se comunican de algún modo entre ellos. También define un programa distribuido como una o más particiones que se ejecutan independientemente en el sistema distribuido. Las particiones se comunican entre ellas intercambiando datos mediante llamadas a procedimientos remotos y objetos distribuidos. En Ada 95 hay dos clases de particiones: activas, que pueden ejecutar en paralelo con otras, posiblemente en un espacio de direcciones separado y probablemente en un computador distinto; y pasivas, que no tienen un hilo de control propio. La comunicación entre particiones activas se realiza de modo estándar utilizando el *Partition Communication Subsystem* (PCS), que tiene una interfaz definida por el lenguaje que se encuentra en el paquete `System.RPC`, de manera que la implementación del PCS pueda ser independiente del compilador y del sistema de ejecución.

El DSA deja algunos aspectos importantes para que sean definidos por la implementación [5][6]. Algunos de ellos son muy importantes a la hora de desarrollar una implementación que tenga características de tiempo real. Por ejemplo, el modo en el que se manejan los RPCs, o las prioridades a las que las tareas debieran ejecutar un RPC son totalmente dependientes de la implementación. Otros aspectos a ser definidos por la implementación como el lenguaje de configuración para describir el sistema distribuido, que puede ser interesante desde una perspectiva de normalización, no son tan importantes desde el punto de vista de tiempo real.

GLADE es la primera implementación con impacto industrial del modelo de programación distribuida de Ada 95. El trabajo [13] propone a GLADE como un

entorno de trabajo para poder desarrollar sistemas distribuidos de tiempo real orientados a objetos. Las referencias [5] y [6] describen los cambios necesarios para realizar una implementación controlable y predecible del DSA, centrándose en el manejo de las prioridades de los manejadores de RPC, y en las restricciones que deben tenerse en cuenta cuando se configura el sistema. Algunos, pero no todos estos aspectos, se han implementado en la versión 3.15p de GLADE.

GLADE se divide en dos partes bien diferenciadas [11]:

- GARLIC: es el PCS, que está compuesto principalmente por el paquete `System.RPC` (siguiendo el Manual de Referencia), y el paquete `System.Garlic` y sus paquetes hijos, que contienen el núcleo del PCS que controla las llamadas en lo relativo a las redes, peticiones concurrentes, localización y arranque de particiones, recuperación y manejo de errores, etc.
- GNATDIST: es la herramienta de particionamiento, responsable de chequear la consistencia del sistema distribuido antes de construirlo llamando a GNAT con los parámetros apropiados para construir los *stubs* necesarios, configurar los filtros que se usen entre las diferentes particiones, enlazar las particiones con GARLIC y construir la secuencia de inicialización y el programa principal que lanzará el sistema distribuido completo en los nodos especificados.

En GARLIC [3], existe un *pool* de tareas manejadoras de RPCs del tipo `RPC_Handler_Task` que se crea en tiempo de inicialización para poder ejecutar de modo concurrente distintas RPCs en una partición dada. Este prealojamiento de tareas se realiza para evitar la sobrecarga que representa la creación y destrucción de tareas para atender a los RPCs. Si estando todas las tareas del *pool* en uso, llegase una nueva RPC, se crearía una nueva tarea para atender a esta RPC. El atributo de partición `Task_Pool` del lenguaje de configuración, permite configurar el *pool* de tareas expresando tres parámetros: el tamaño mínimo del *pool* (número de manejadores de RPC prealojados y siempre disponibles), una cota intermedia para este *pool* (cuando una RPC se completa, la tarea manejadora de ese RPC se elimina si el número de tareas del *pool* es mayor que ese techo), y el tamaño máximo del *pool* (es un límite del número de llamadas remotas simultáneas; si el número de llamadas remotas activas es mayor que este número, la petición queda pendiente hasta que una tarea manejadora de RPCs queda libre).

GLADE también utiliza alojamiento dinámico de tareas cuando llega un mensaje a la partición receptora de un RPC. Cada partición tiene uno o más puertos de entrada TPC/IP (creados a partir de parámetros de configuración como el pragma `Boot_Location`, o el atributo de partición `Self_Location`), y para cada puerto de entrada se crea en tiempo de inicialización una nueva tarea del tipo `Accept_Handler`, que espera la recepción de mensajes. Cuando llega un mensaje, esta tarea cede el procesado del mensaje a otra tarea del tipo `Connect_Handler`, que se encuentra en un segundo *pool* de tareas. En tiempo de inicialización, este *pool* se encuentra vacío y la tarea se crea dinámicamente en caso de que sea necesario. La tarea `Connect_Handler` se encarga del procesado del mensaje y de llamar a la tarea manejadora de RPCs que ejecutará el subprograma, dejando libre al

`Accept_Handler` para esperar nuevos mensajes entrantes en el puerto de recepción. Aunque es posible alojar estáticamente las tareas manejadoras de RPCs, la creación de las tareas `Connect_Handler` es dinámica y no es apropiada para aplicaciones de tiempo real. Se necesita un mecanismo para evitar el alojamiento dinámico de tareas internas.

El DSA no tiene modo de expresar la prioridad a la que se debería ejecutar la tarea que sirve una RPC, ni las prioridades de los mensajes en la red de comunicaciones. En la última versión pública de GLADE (3.15p) se introduce el concepto de política de prioridad que contempla dos posibles valores: `Client_Propagated` y `Server_Declared`. EL primer valor se corresponde con el de las anteriores versiones de GLADE, en el que la prioridad de la tarea que invoca el RPC se codifica en el mensaje mandado a la partición receptora, para que la tarea que ejecuta el RPC lea la prioridad de la tarea invocante (que está codificada en los datos recibidos), y fije su propia prioridad a dicho valor. En cuanto al segundo valor, establece una prioridad fija para todas las tareas manejadoras de RPC dentro de una misma partición. Esta prioridad se fija mediante el atributo de partición denominado `Priority` mientras que la política de prioridad usada se establece mediante el pragma de configuración de mismo nombre que el atributo (`Priority`). Todas las particiones utilizarán la misma política de prioridad, la establecida por el pragma en el archivo de configuración. La prioridad original de las tareas manejadoras de RPCs se fija por defecto a su valor máximo volviendo a dicho valor una vez ejecutado el RPC. En [5] mostramos que era posible mejorar el comportamiento de tiempo real si la aplicación pudiese especificar la prioridad inicial del RPC así como la prioridad de cada llamada. Además, GLADE no soporta el que se puedan establecer prioridades para los mensajes enviados a la red. La Figura 1 muestra la arquitectura de GLADE y el esquema de prioridades que utiliza.

En GLADE las llamadas entre dos particiones situadas en el mismo nodo procesador se realizan utilizando las capacidades de la red, aunque sería más eficiente hacerlo utilizando algún tipo de protocolo local que evitara el paso a través de la red. Esta solución sería además más portable, pues se podrían utilizar redes que por criterio de optimización no permitieran la transmisión de un nudo hacia sí mismo.

Hay algunos aspectos en GLADE que introducen sobrecargas no uniformes que deberían minimizarse en sistemas de tiempo real estricto. Por ejemplo, cuando una tarea de la partición llamante haya mandado un mensaje a la red para realizar una RPC, espera la respuesta en un único *entry* de un objeto protegido. Cada tarea de la partición llamante se encolará en el mismo *entry*, de modo que cuando se procese la respuesta a la llamada, se necesita determinar quien es el "propietario" de la respuesta. Para llevarlo a cabo, cada tarea encolada en el *entry* es desencolada y encolada de nuevo salvo que sea aquella a la que estaba dirigida la respuesta. Esto implica una sobrecarga innecesaria proporcional al número de tareas que esperan respuesta de una RPC.

El pragma `Shared_Passive` definido en el DSA se utiliza para manejar datos globales compartidos entre particiones activas. La implementación de GLADE de este pragma se basa en el uso del sistema de ficheros para soportar los datos

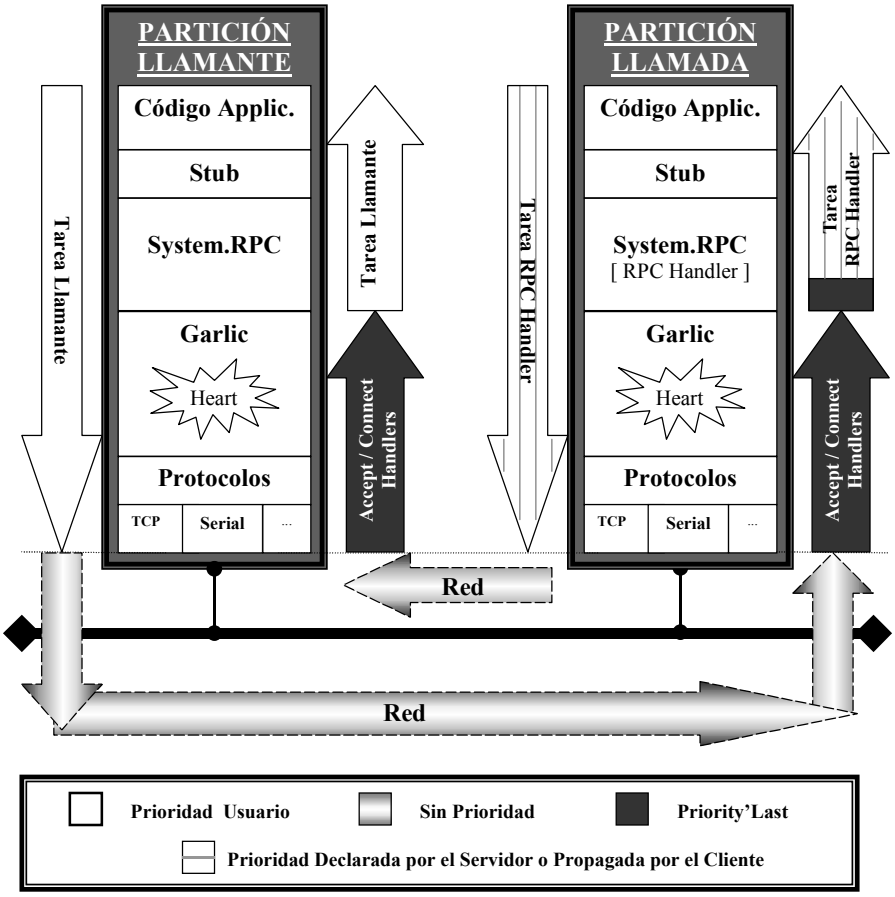

Figura 1 Esquema de prioridades en la arquitectura GLADE

correspondientes a particiones pasivas, y asume que los datos serán compartidos utilizando algún tipo de sistema de ficheros de red. Sin embargo, la mayoría de las implementaciones de este tipo de sistemas no tiene comportamiento de tiempo real, y por tanto no proporcionaremos soporte para este pragma en RT-GLADE.

3 Características de RT-GLADE

Las modificaciones que se han hecho a GLADE para soportar comportamiento de tiempo real se centran principalmente en los siguientes aspectos que serán descritos separadamente con más detalle:

- Priorización de la tarea manejadora de RPCs y de los mensajes a través de la red de acuerdo a las recomendaciones dadas en [5].
- Incorporación de una red de comunicaciones basada en prioridades sobre Ethernet estándar, y de un nuevo protocolo local para incrementar la eficiencia de las llamadas hechas en el mismo nodo de procesamiento.

- Eliminación de la creación dinámica de tareas a la recepción de las RPCs, y mejora del mecanismo de espera a la respuesta de las RPCs en la partición llamante.
- Adaptación de la configuración del sistema a las nuevas capacidades, incluyendo los aspectos relacionados con las redes de comunicación de tiempo real.

La modificación propuesta hace que la implementación se mantenga conforme al actual DSA, aunque se necesitan algunas nuevas interfaces para soportar las características de tiempo real.

3.1. Control de la prioridad sobre todo el sistema

Para poder controlar el comportamiento temporal de una aplicación de tiempo real se necesita algún mecanismo para especificar las prioridades con las que se ejecutarán las RPCs, así como para los mensajes que se enviarán a través de la red.

De acuerdo con lo discutido en [6], usamos el tipo `Global_Priority` definido en el paquete `System.Garlic.Priorities` que representa un valor con significado global a lo largo de todo el sistema distribuido, y lo modificamos para hacerlo puro porque es necesario intercambiar prioridades a través de diferentes particiones. También usamos las funciones de mapeado definidas en `System.Garlic.Priorities.Mapping` que traducen los valores de este tipo a valores del tipo `System.Priority`. El mismo esquema de nombres se utiliza para añadir nuevas funciones de mapeado entre el tipo `Global_Priority` y las prioridades de la red RT-EP que utilizamos en nuestra implementación.

Además, creamos el paquete `RPC_Priorities`, que contiene las operaciones para fijar las prioridades de los mensajes salientes que son enviados por la partición llamante de la RPC, la prioridad de la RPC para esa llamada en particular, y para el mensaje entrante que es devuelto por la partición llamada cuando la ejecución de la RPC haya terminado:

```
with System.Garlic.Priorities;
use System.Garlic.Priorities;
package RPC_Priorities is
   procedure Set
      (RPC_Handler : in Global_Priority);
   procedure Set
      (RPC_Handler,
       Outgoing_Message: in Global_Priority);
   procedure Set
      (RPC_Handler,
       Outgoing_Message,
       Incoming_Message: in Global_Priority);
   procedure Get
      (RPC_Handler,
       Outgoing_Message,
       Incoming_Message: out Global_Priority);
end RPC_Priorities;
```

El procedimiento `Set` se utiliza para fijar las prioridades usadas por las RPCs y APCs futuros llevados a cabo por la tarea llamante y pueden ser invocados por el usuario antes de hacer la llamada. Estas prioridades tienen efecto hasta que se vuelva a llamar al procedimiento `Set`. De este modo, la aplicación puede especificar las prioridades de sus RPCs individualmente o agrupando las llamadas utilizando las mismas prioridades. Los valores iniciales para las prioridades se establecen a un nivel intermedio. El procedimiento `Get` devuelve el valor actual de los valores de las prioridades de la RPC. La implementación de este paquete almacena las prioridades creando tres atributos de tarea utilizando las características descritas en el paquete opcional pero estándar `Ada.Task_Attributes` [5].

Para evitar cambios de contexto, hemos trasladado el trabajo realizado por las tareas `Connect_Handler` a las tareas `Accept_Handler`. Como el subsistema de comunicaciones implementa encolado de mensajes, no hay posibilidad de que se pierdan. Para evitar posibles inversiones de prioridad causadas por la prioridad inicial de las tareas manejadoras de RPCs del pool, permitimos su configuración mediante el atributo de partición llamado `Priority`, que especifica el tipo `Global_Priority`. En nuestra implementación utilizamos este atributo también para establecer la prioridad de las tareas `Accept_Handler`.

La Figura 2 muestra una RPC típica y cómo se manejan las prioridades utilizando RT-GLADE. En el nivel de aplicación, la tarea llamante fija los valores de prioridad involucrados en la llamada: del mensaje de salida, del mensaje de respuesta, y de la ejecución remota en la partición llamada. Entonces ejecuta todo el código a su prioridad hasta que manda la petición a través de la red de tiempo real. Cuando la tarea llamante alcanza las funciones `Do_RPC` o `Do_APC` del paquete `System.RPC`, en vez de escribir la prioridad de la partición llamante en el mensaje tal y como ocurre en GLADE, llamamos a `RPC_Priorities.Get` y escribimos en el mensaje las prioridades `RPC_Handler` e `Incoming_Message`. La prioridad a la que se envía el mensaje a través de la red es el valor de `Outgoing_Message` (tras ser mapeada a su valor apropiado).

Cuando el mensaje llega a la partición llamada, una tarea `Accept_Handler` (ver subsección 3.3) ejecutándose a la prioridad establecida procesa el mensaje. Como parte de este procesado, lee la prioridad del correspondiente manejador de RPC del mensaje, selecciona un manejador libre del *pool*, y fija su prioridad al valor deseado y despierta al manejador pasándole el mensaje. Esta estrategia evita cambios de contexto comparado con la implementación original de GLADE en la que era el propio manejador de RPC quien fijaba su prioridad.

La tarea manejadora de RPC lee el valor de prioridad `Incoming_Message` y los parámetros de la llamada del mensaje e invoca el procedimiento remoto implicado en el RPC. Cuando la tarea se completa, los parámetros de vuelta y los indicadores de error, si los hay, se mandan a la partición llamante a la prioridad `Incoming_Message`. Entonces la tarea manejadora de RPC se suspende volviendo al *pool*. Su prioridad no se cambia debido a que antes de que empiece a ejecutarse una tarea `Accept_Handler` fijará su prioridad al valor apropiado para el nuevo RPC.

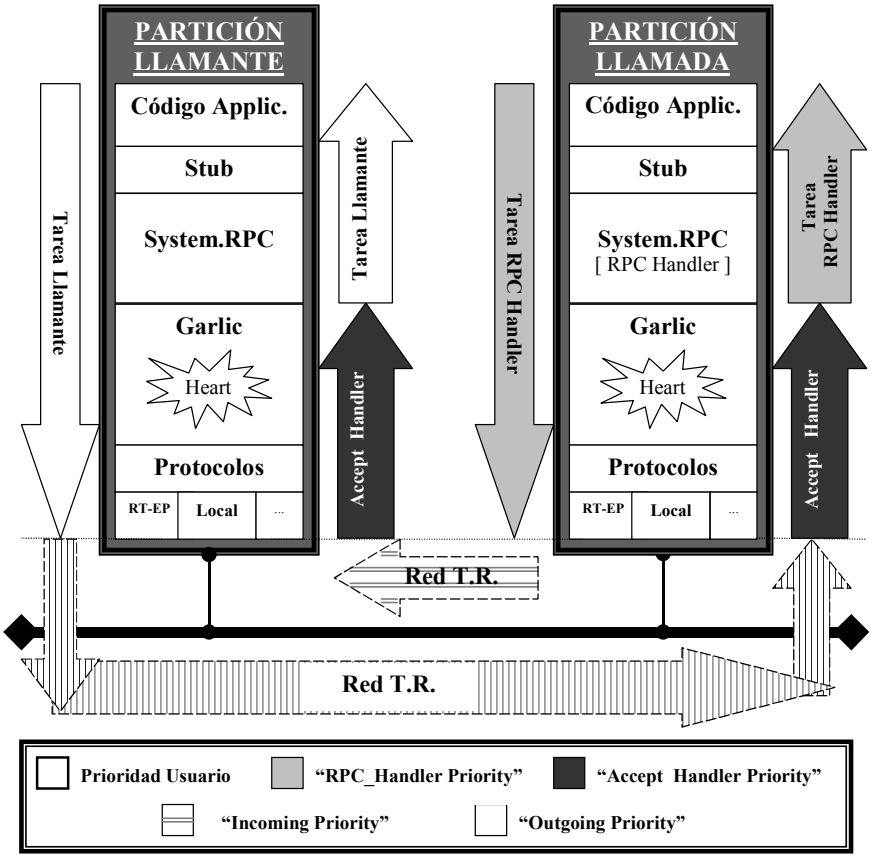

Figura 2 Esquema de prioridades en la arquitectura RT-GLADE

Cuando el mensaje con los parámetros de retorno llega a la partición llamante, se procesa a la prioridad establecida por una tarea `Accept_Handler` (ver subsección 3.3) que pasa los datos a la tarea llamante que estaba suspendida esperando la respuesta.

3.2. Nuevas capacidades de comunicación añadidas a GARLIC

La principal característica de RT-GLADE es que permite al usuario tener un completo control de las prioridades involucradas en la ejecución y en la comunicación de las RPCs, de modo que se puede hacer un análisis de planificabilidad completo de una aplicación distribuida. Como RT-GLADE usa una red, es obvio que es necesario usar una con características de tiempo real. La red basada en TCP/IP proporcionada por GLADE no es adecuada para comunicaciones de tiempo real. En nuestra implementación de RT-GLADE usamos la red RT-EP (Real-Time Ethernet Protocol) [8], basada en un paso de testigo codificado sobre protocolo Ethernet para comunicaciones multipunto en aplicaciones de tiempo real utilizando hardware

Ethernet estándar. Este protocolo puede manejar mensajes de una longitud de hasta 1492 bytes en un único paquete. Actualmente RT-GLADE puede utilizar una capa de particionamiento de mensajes en paquetes, por lo que no hay restricción en la longitud de los mensajes que se generan para utilizar este protocolo.

En la implementación pueden utilizarse diversos protocolos, así pues, cuando se ejecuta una RPC, se necesita determinar el protocolo que se va a usar para la comunicación. Una vez que se ha seleccionado el protocolo, se necesita pasar la información relevante para contactar con la partición remota a través de la red. Esta información comprende la identificación del nodo donde se encuentra la partición y cualquier otra posible información para el protocolo elegido, por ejemplo el puerto de destino, como se hace en la implementación de GLADE.

RT-EP [8] utiliza un concepto similar al de puerto denominado *canal de recepción*, que es un número que identifica los destinos de comunicación en una estación determinada. Para poder transmitir por RT-EP es necesario conocer la dirección MAC del nodo de destino y el canal de recepción de ese nodo. Esta información se codifica utilizando el atributo de partición `Self_Location` y el pragma `Boot_Location` del lenguaje de configuración (ver subsección 3.4).

Hemos definido un nuevo protocolo virtual al que llamamos protocolo Local, que se utiliza para mejorar la eficiencia cuando se realiza una llamada a un subprograma perteneciente a una unidad de biblioteca RCI, y que está situada en una partición localizada en el mismo nodo de procesamiento que la partición llamante. El uso del protocolo Local también evita la limitación de RT-EP de no permitir mandar un mensaje al mismo nodo. Este protocolo se implementa con la misma interfaz que RT-EP [8], y también utilizan una cola de prioridad de recepción para cada canal de recepción definido en el proceso de configuración. Un mensaje enviado a través del protocolo Local se encola con la prioridad de la tarea llamada, que es la prioridad de la tarea manejadora de RPC para la petición, y a la prioridad de la tarea llamante para la respuesta. Cuando un mensaje se envía a través del protocolo Local, ni la prioridad `Outgoing_Message` ni la `Incoming_Message` se utilizan debido a que no hay uso de red. Internamente, la prioridad de la tarea llamante se codifica en el mensaje para encolar la respuesta en el orden correcto.

Una vez que una tarea llamante que ejecuta una RPC ha comprobado que la llamada no pertenece a la misma partición ni a una partición localizada en el mismo nodo, la implementación debe determinar cómo contactar con la partición remota, es decir, qué protocolo usar. Para ello, la tarea llamante chequea una tabla que contiene la información del protocolo a utilizar para acceder a la partición deseada. No es necesario que si, por ejemplo, una partición P1 tiene que contactar con una partición P2 utilizando el protocolo RT-EP, la partición P2 tenga que usar el mismo protocolo para contactar con P1. En el proceso de configuración utilizamos los *Partition Links* (ver subsección 3.4) para seleccionar el protocolo a usar entre la partición llamante P1 y la partición llamada P2.

3.3. Gestión de la recepción de RPCs

RT-GLADE elimina de la implementación de GLADE la creación dinámica de tareas para la ejecución de RPCs, configurando el número de tareas manejadoras de RPCs en el *pool* con un número suficiente de tareas estáticas, asegurando así la ejecución de todas las posibles RPCs concurrentes. Para ello, fijamos el número mínimo de tareas en el *pool* utilizando el atributo `Task_Pool`. Si queremos un *pool* de tareas estático, los otros dos números de este atributo (la cota intermedia y el máximo) se deben fijar al mismo valor. Esto sigue las recomendaciones dadas en [6].

El otro lugar en el que GLADE crea tareas dinámicamente es en la gestión de las tareas `Accept_Handler` y `Connect_Handler`. En RT-GLADE cada partición tiene una única tarea `Accept_Handler` (no tiene `Connect_Handler`) por canal de recepción configurado en el protocolo de tiempo real (actualmente el protocolo Local y RT-EP). Estas tareas se crean en tiempo de inicialización y se bloquean inmediatamente a la espera de recibir mensajes de sus respectivos canales de recepción.

Para evitar la sobrecarga no uniforme asociada con el uso de un único *entry* para esperar las respuestas de RPC, proponemos utilizar una familia de *entries*. Cada tarea que espera una respuesta a una RPC se encola en un *entry* diferente. Antes de enviar el mensaje, determinamos el identificador del *entry* en el que la tarea llamante será encolada. Este identificador se manda en el mensaje y se vuelve a enviar en el mensaje de vuelta junto con la respuesta para determinar qué *entry* deber ser servido. Después de usarlo, el *entry* vuelve a estar libre para ser usado por otra tarea llamante. Se añade un nuevo parámetro de configuración para especificar el número de *entries* del objeto protegido.

3.4. Parámetros de configuración en RT-GLADE

La herramienta GNATDIST posee su propia interfaz para poder describir la configuración de una aplicación distribuida. Esta herramienta lee un fichero ".cfg" que se rige según las reglas de su propio lenguaje de configuración y cuyo estilo es parecido al de Ada. RT-GLADE utiliza un subconjunto de este lenguaje de configuración definido en GLADE junto con algunos añadidos que se han realizado expresamente en nuestra implementación. A continuación se indicarán cuáles de los atributos y pragmas que se utilizan en GLADE son especialmente relevantes y cuáles no se utilizan o están prohibidos:

- Para la configuración y declaración de particiones así como para establecer cuál es el procedimiento principal dentro de una partición, se utiliza el mismo esquema que en GLADE.
- El valor que debe establecerse para el pragma `Starter` es `None`, porque las diversas particiones que se van a generar queremos arrancarlas manualmente, sin utilizar la *shell* remota convencional usada por GLADE. Esto se debe a que RT-GLADE utiliza el sistema operativo de tiempo real MaRTE OS. Debido a su entorno cruzado de desarrollo, las particiones se generarán en un *host* a partir del

que los diversos nodos que constituyan la aplicación distribuida cargarán la partición que les corresponda.

- Por lo comentado en el punto anterior, no es necesario fijar el atributo de partición `Host` para cada una de las particiones.
- El atributo de partición `Self_Location` se utiliza para determinar cómo una partición puede contactar con otra. La información que se codifica en este atributo consta de: un nombre de protocolo, un identificador de nodo procesador, un identificador de canal de recepción y una lista de particiones para fijar los correspondientes *Partition Links*. Un *Partition Link* establece cuál es el protocolo preferido para que las particiones definidas en dicha lista contacten con la partición a la que se le aplicó el atributo `Self_Location`.

  También debe especificarse el pragma `Boot_Location` para determinar la localización del "*boot server*" en el que deben registrarse las particiones. El "*boot server*" se encuentra en la partición principal. La información que debe proporcionarse a este atributo es la misma que la comentada para el atributo `Self_Location`.

  Para no saturar de información poco relevante la configuración de la aplicación distribuida, así como para proporcionar una mayor facilidad de comprensión del archivo de configuración, se ha preferido que cierta información relativa a los protocolos de comunicación se sitúe en otro archivo. Este archivo proporcionará la relación entre los identificadores de los nodos y las direcciones de contacto de los mismos, por ejemplo la dirección MAC asociada a cada nodo si se utilizase RT-EP.
- El atributo de partición `Task_Pool` debe configurarse tal y como se ha descrito en la subsección 3.3.
- No son necesarios ni el servicio de filtrado ni los canales bidireccionales que define GLADE y que están principalmente enfocados a este servicio.
- Se prohibe el uso de los pragmas y de los atributos relacionados con la configuración de particiones pasivas. Es necesario realizar un mayor esfuerzo en este aspecto para poder aportar una solución válida que conlleve características de tiempo real.
- También están prohibidos para aplicaciones de tiempo real estricto, los aspectos dinámicos relacionados con la reconexión en GLADE debido a su comportamiento temporal impredecible, aunque podrían utilizarse en una futura implementación para su uso en sistemas con características de tiempo real no estrictas.
- El atributo de partición `Priority` se utiliza para fijar la prioridad a la que deben ejecutarse las tareas `Accept_Handler`.
- Se ha añadido un nuevo atributo de partición denominado `Max_RPC_Replies` para fijar el número de *entries* del objeto protegido en el que las tareas que han realizado una RPC esperan su respuesta.

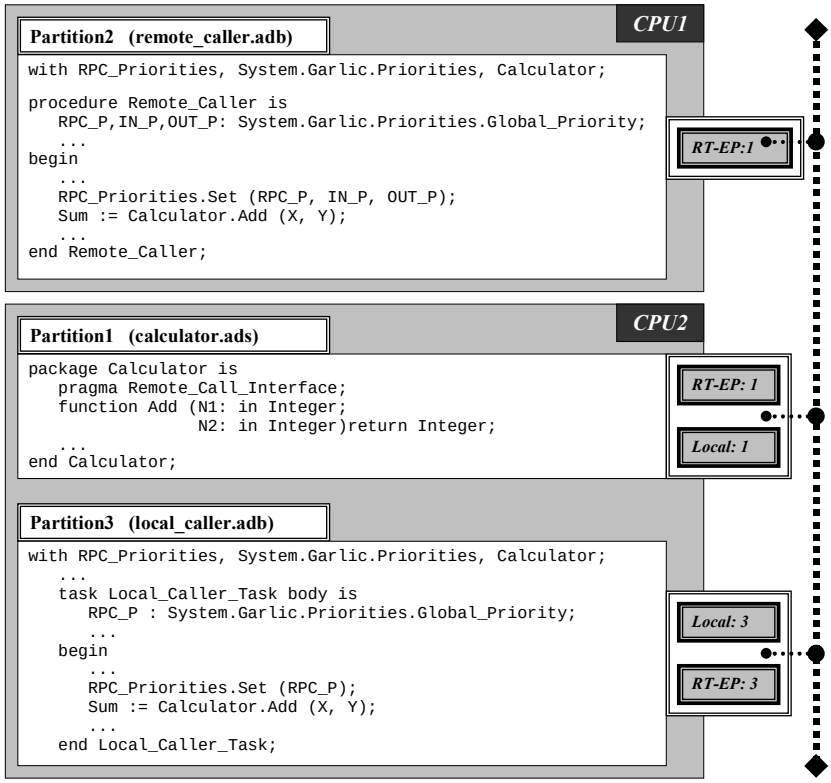

Figura 3 Ejemplo sencillo utilizando RT-GLADE

4 Ejemplo de uso

La Figura 3 muestra un ejemplo sencillo de una aplicación distribuida con dos CPUs conectadas por una red que utiliza el protocolo RT-EP. El código del ejemplo se divide en tres unidades de biblioteca: el procedimiento principal `Remote_Caller`, el paquete RCI `Calculator`, y la unidad que contiene la tarea `Local_Caller_Task`. Cada unidad de biblioteca se aloja en una partición diferente y se asigna a una CPU tal como se muestra. El procedimiento principal hace una RPC a la función `Calculator.Add` a través del protocolo RT-EP. Esta llamada debe especificar las tres prioridades necesarias para hacer una RPC. La tarea `Local_Caller_Task` llama a la misma función, pero debido a que está localizada en la misma CPU utilizará el protocolo Local, y por tanto sólo debe especificarse la prioridad `RPC_Handler`.

El fichero de configuración RT-GLADE en el que se ve la distribución de las particiones y la manera en la que se utilizan los protocolos es el siguiente:

```
configuration Configuration_File is
   pragma Starter (None);
   Partition1: partition := (Calculator);
```

```
   Partition2: partition := (Remote_Caller);
   Partition3: partition := (Local_Caller);
   procedure Remote_Caller is in Partition2;
   pragma Boot_Location (("RT_EP","CPU1:1"));
   for Partition1'Self_Location use
      ((("RT_EP","CPU2:1"), ("Local","CPU2:1:Partition3")));
   for Partition2'Self_Location use
      (("RT_EP","CPU1:1"));
   for Partition3'Self_Location use
      ((("RT_EP","CPU2:3"), ("Local","CPU2:3:Partition1")));
   for Partition1'Priority use 27;
   for Partition1'Task_Pool use (8, 8, 8);
   for Partition1'Max_RPC_Replies use 4;
end Configuration_File;
```

El atributo de partición `Self_Location` aplicado a `Partition2` determina que tiene dos protocolos de comunicaciones (RT-EP y Local) y los canales utilizados. Aunque en este caso no es necesario especificar *Partition Links* porque sólo existe una red disponible, la `Partition2` explícitamente requiere de la `Partition1`: "si me envías un mensaje tienes que utilizar el protocolo RT-EP y el canal de recepción 3". En este ejemplo, se puede ver también que el número de tareas manejadora de RPC en la `Partition1` se establece estáticamente al valor 8, la prioridad inicial de estas tareas y de los Accept_Handler a 27, y el número máximo de respuestas pendientes simultáneas a 4.

5 Evaluación

El modelado de una aplicación que use RT-GLADE se puede hacer utilizando los modelos descritos en [8] para RT-EP y [9] para las RPCs. Las medidas de los parámetros temporales especificados en el modelo dependen de la plataforma particular, por lo que sus valores absolutos no son representativos. En esta sección mostraremos los tiempos de respuesta de un ejemplo sencillo en el que se verá cómo el control total de las prioridades de cada RPC y de los mensajes permite obtener mejores resultados que con los dos modelos soportados por el GLADE original. La Figura 4 muestra la arquitectura de esta aplicación sencilla con dos procesadores y cinco tareas (numeradas de 1 a 5) que ejecutan operaciones remotas (denotadas como ROP 1 a 5) en el otro procesador. Los tiempos de ejecución de peor caso (C) y los periodos (T) también se muestran en la figura. La aplicación se divide en dos particiones, una por cada procesador. Las tareas `Task_1` y `Task_2` tienen requisitos temporales estrictos con plazos iguales a sus respectivos periodos; el resto de tareas no tienen requisitos de tiempo real, por lo que ejecutan a un nivel de prioridad más bajo. Las unidades de tiempo expresadas son segundos.

Se han realizado medidas sobre los esquemas de prioridad propuestos por GLADE y el esquema de asignación propuesto por la herramienta MAST [4] (que no es posible de implementar en GLADE). Para comprobar estos esquemas se utiliza la plataforma RT-GLADE/MaRTE OS, y se emulan los esquemas propuestos por GLADE. Los resultados se muestran en la Tabla 1. Dado que el GLADE original no soporta

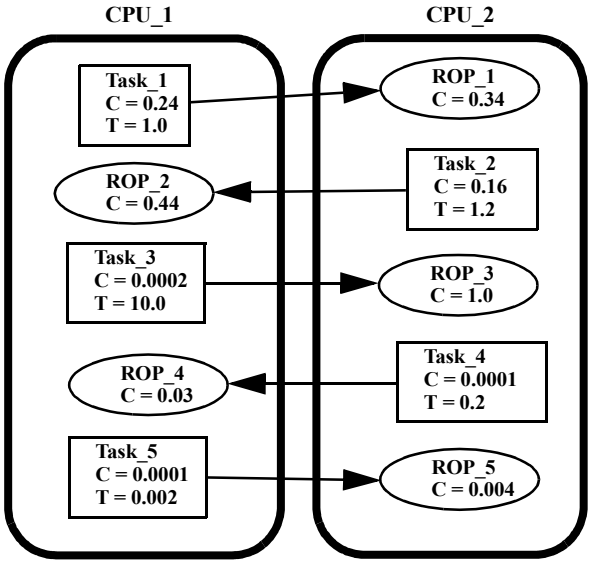

Figura 4 Arquitectura de una aplicación de ejemplo

prioridades en la red, hemos utilizado un único nivel de prioridad para todos los mensajes de la red. Las medidas se han realizado en la misma plataforma, por lo que las diferencias se deben únicamente a los diferentes esquemas de prioridad empleados. En esta tabla, por tanto, se muestran los tiempos de respuesta de peor caso para las tareas `Task_1` y `Task_2` con la asignación de prioridades MAST (esta asignación sólo es posible en RT-GLADE), y con las dos posibles políticas de GLADE: `Client_Propagated` y `Server_Declared`, cada una con diferentes combinaciones de prioridades.

Tabla 1. Comparación de los diferentes esquemas de prioridad RT-GLADE/MaRTE OS

Esquema de Prioridad	Task_1 Prio.	ROP_1 Prio.	Task_2 Prio.	ROP_2 Prio.	Task_1 WCRT (s)	Task_2 WCRT (s)
MAST	Alta	Media	Alta	Media	0.806	0.898
Client_Propagated	Alta	Alta	Media	Media	0.638	1.527
	Media	Media	Alta	Alta	No acotado	0.653
Server_Declared	Alta	Media	Alta	Media	1.538	1.149
	Alta	Alta	Media	Media	2.148	2.887
	Media	Media	Alta	Alta	No acotado	0.655

Como se puede observar los requisitos temporales de este ejemplo en particular solamente se cumplen bajo el esquema de prioridades posible en RT-GLADE. Los resultados del esquema `Server_Declared` son especialmente malos porque las operaciones remotas `ROP_3`, `ROP_4` y `ROP_5` invocadas por una tarea de baja prioridad se ejecutan forzosamente a la misma prioridad que las invocaciones de prioridades alta o media `ROP_1` y `ROP_2`. Los tiempos de respuesta no acotados se obtienen en algunos

casos porque la tarea es incapaz de completar su trabajo antes de su periodo y el trabajo continúa acumulándose dando lugar a tiempos de respuesta crecientes.

Tabla 2. Comparación de los diferentes esquemas de prioridad GLADE/LINUX

Esquema de Prioridad	Task_1 Prio.	ROP_1 Prio.	Task_2 Prio.	ROP_2 Prio.	Task_1 WCRT (s)	Task_2 WCRT (s)
Client_Propagated	Alta	Alta	Media	Media	1.389	1.496
	Media	Media	Alta	Alta	1.293	1.416
Server_Declared	Alta	Media	Alta	Media	1.526	1.336
	Alta	Alta	Media	Media	1.523	1.337
	Media	Media	Alta	Alta	1.440	1.394

La Tabla 2 muestra los tiempos de respuesta de peor caso para las tareas `Task_1` y `Task_2` ejecutando sobre GLADE/LINUX. Se puede observar que los tiempos siempre convergen, pero en ninguno de los casos se cumplen los requisitos temporales impuestos. De hecho los tiempos bajo los diferentes esquemas de prioridad son muy parecidos debido al tratamiento que hace LINUX de las prioridades de las tareas.

6 Conclusiones

Se ha presentado RT-GLADE, una versión con características de tiempo real de GLADE, la implementación de GNAT del Anexo de Sistemas Distribuidos de Ada 95. La nueva implementación continúa siendo conforme al DSA y se ofrece como software libre bajo licencia GNU. Para mejorar la controlabilidad y predecibilidad del comportamiento temporal se ha modificado el esquema de prioridades de GLADE para que la aplicación pueda controlar totalmente las prioridades de los mensajes de comunicación, así como de los manejadores de RPCs. También se ha eliminado la necesidad de crear tareas dinámicamente y algunas fuentes de sobrecarga no uniforme.

Además de cambios en el entorno (middleware) se han añadido servicios de red capaces de proporcionar comportamiento de tiempo real. Se ha utilizado RT-EP, un protocolo de tiempo real basado en hardware Ethernet estándar, y por razones de eficiencia, se ha añadido un protocolo Local para las llamadas dirigidas al mismo nodo procesador.

En resumen, utilizando RT-GLADE se pueden construir aplicaciones que garanticen el cumplimiento de sus requisitos de tiempo real. Como conclusión, hemos mostrado cómo es posible desarrollar sistemas distribuidos de tiempo real haciendo uso de los servicios del Ada, lo que le puede ayudar a mantenerse como lenguaje de referencia de los sistemas de tiempo real.

Bibliografía

[1] Ada-Core Technologies, Ada 95 GNAT Pro, http:// www.gnat.com/

[2] M. Aldea and M. González. “MaRTE OS: An Ada Kernel for Real-Time Embedded Applications”. Proceedings of the International Conference on Reliable Software

Technologies, Ada-Europe 2001, Leuven, Belgium, in Lecture Notes in Computer Science, LNCS 2043, May 2001.

[3] Y. Kermarrec, L. Pautet, and S. Tardieu. “GARLIC: Generic Ada Reusable Library for Interpartition Communication”. Proceedings of Tri-Ada’95, Anaheim, California, USA, ACM, 1995.

[4] M. González Harbour, J.J. Gutiérrez, J.C. Palencia and J.M. Drake. “MAST: Modeling and Analysis Suite for Real-Time Applications”. Proceedings of the Euromicro Conference on Real-Time Systems, Delft, The Netherlands, June 2001.

[5] J.J. Gutiérrez García, and M. González Harbour. “Prioritizing Remote Procedure Calls in Ada Distributed Systems”. Proceedings of the 9th International Real-Time Ada Workshop, ACM Ada Letters, XIX, 2, pp. 67–72, June 1999.

[6] J.J. Gutiérrez García, and M. González Harbour. “Towards a Real-Time Distributed Systems Annex in Ada”. Proceedings of the 10th International Real-Time Ada Workshop, ACM Ada Letters, XXI, 1, pp. 62–66, March 2001.

[7] Jane W. S. Liu. “Real-Time Systems”. Prentice Hall, 2000.

[8] J.M. Martínez, M. González Harbour, and J.J. Gutiérrez. “RT-EP: Real-Time Ethernet Protocol for Analyzable Distributed Applications on a Minimum Real-Time POSIX Kernel”. Proceedings of the 2nd International Workshop on Real-Time LANs in the Internet Age, RTLIA 2003, Porto (Portugal), July 2003.

[9] J. Javier Gutiérrez, José M. Drake, Michael González Harbour, and Julio L. Medina. “Modeling and Schedulability Analysis in the Development of Real-Time Distributed Ada Systems”. Proceedings of the 11th International Real-Time Ada Workshop, ACM Ada Letters, Vol. XXII, No. 4, pp. 58–65, December 2002.

[10] Object Management Group, “Realtime CORBA Joint Revised Submission”. OMG Document orbos/99-02-12 ed., March 1999.

[11] L. Pautet and S. Tardieu, “Inside the Distributed Systems Annex”. Proceeding of the Intl. Conf. on Reliable Software Technologies, Ada-Europe’98, Uppsala, Sweden, in LNCS 1411, Springer, pp. 65–77, June 1998.

[12] L. Pautet, T. Quinot, and S. Tardieu. “CORBA & DSA: Divorce or Marriage”. Proc. of the International Conference on Reliable Software Technologies, Ada-Europe’99, Santander, Spain, in Lecture Notes in Computer Science No. 1622, pp. 211–225, June 1999.

[13] L. Pautet and S. Tardieu. “GLADE: a Framework for Building Large Object-Oriented Real-Time Distributed Systems”. Proc. of the 3rd IEEE Intl. Symposium on Object-Oriented Real-Time Distributed Computing, (ISORC'00), Newport Beach, USA, March 2000.

[14] IEEE Std. 1003.13-2003. Information Technology -Standardized Application Environment Profile- POSIX Realtime and Embedded Application Support (AEP). The Institute of Electrical and Electronics Engineers.

[15] Scott Moody. “Object-Oriented Real-Time Systems Using a Hybrid Distributed Model of Ada 95's Built-in DSA Capability (Distributed Systems Annex-E) and CORBA”. Proceedings of the 8th International Real-Time Ada Workshop, ACM Ada-Letters, XVII, 5, pp. 71–76, September-October 1997.

[16] S. Tucker Taft, and R.A. Duff (Eds.). “Ada 95 Reference Manual. Language and Standard Libraries”. International Standard ISO/IEC 8652:1995(E), in LNCS 1246, Springer, 1997.