CORBA & DSA: Análisis y evaluación de sus implementaciones desde la perspectiva de los sistemas de tiempo real

Héctor Pérez Tijero
Grupo de Computadores y Tiempo Real
Universidad de Cantabria
39005 Santander, SPAIN
perezh@unican.es

J. Javier Gutiérrez
Grupo de Computadores y Tiempo Real
Universidad de Cantabria
39005 Santander, SPAIN
gutierjj@unican.es

Resumen

En este trabajo se presenta un estudio de dos aproximaciones estándares al *middleware* (o software de intermediación) para distribución y la problemática asociada a sus implementaciones para sistemas de tiempo real. Los estándares son CORBA y su especificación de tiempo real RT-CORBA, y el anexo para sistemas distribuidos o DSA del lenguaje de programación Ada. El estudio se centra en la comparación de la funcionalidad que ofrecen para la implementación de aplicaciones distribuidas de tiempo real, en el análisis de sus implementaciones desde el punto de vista de la gestión de las llamadas a recursos remotos, y en la evaluación de los tiempos de respuesta que se pueden obtener en las llamadas remotas para dar una idea de las sobrecargas introducidas.

1. Introducción

El concepto de aplicación distribuida no es nuevo, existe desde el momento en que se pueden comunicar dos computadores. Sin embargo, las técnicas de programación de estos sistemas ha evolucionado fuertemente y están consiguiendo una gran relevancia especialmente en la última década. Desde los mecanismos tradicionales de paso de mensajes para la comunicación entre partes de una aplicación distribuida, en los que las comunicaciones entre partes de la aplicación las hacía de forma explícita el programador, se ha llegado a las técnicas de distribución de objetos pasando por las llamadas a procedimientos remotos (RPCs, *Remote Procedure Calls*). En éstas, el uso de las operaciones se hace de forma transparente al hecho de que la funcionalidad sea ofrecida en el propio procesador o en uno remoto.

El paradigma de distribución de objetos es probablemente el que más relevancia tiene en este momento en las aplicaciones industriales, y está representado fundamentalmente por el estándar CORBA [11]. Este estándar incluye también otras técnicas de distribución de más alto nivel como el modelo de componentes (CCM, *CORBA Component Model*), pero el calado actual en la industria es menor. CORBA proporciona un mecanismo de distribución de objetos basado en un lenguaje propio de especificación de interfaces (IDL, *Interface Definition Language*) que permite el uso de diferentes lenguajes de programación en el desarrollo de una aplicación.

Por otra parte, se encuentran los lenguajes de programación que permiten el desarrollo de aplicaciones distribuidas. Este es el caso del lenguaje Java (estándar *de facto*) con su especificación para sistemas distribuidos Java RMI (*Java Remote Method Invocation*) [16] que se basa en la distribución de objetos, o del lenguaje estándar Ada con su DSA (*Distributed Systems Annex*, Annex E) [18] que soporta tanto la distribución de objetos, como las RPCs.

En este trabajo nos vamos a centrar en las perspectivas de tiempo real de la distribución con los estándares CORBA y Ada. No contemplamos Java por el hecho de que aún no tiene del todo resueltos sus aspectos básicos para tiempo real. RT-CORBA [12] ofrece la especificación de CORBA para sistemas de tiempo real, y aunque el DSA de Ada no está especialmente especificado para sistemas de tiempo real existen trabajos que demuestran que es

posible hacer implementaciones de tiempo real dentro del estándar [13][6][7].

En nuestro grupo de investigación llevamos varios años trabajando con sistemas distribuidos de tiempo real programados en Ada. Creemos que aunque el DSA no es muy popular permite programar sistemas distribuidos de una forma sencilla, y puede incorporar en el middleware de tiempo real conceptos como la transacción distribuida [7] y la planificación flexible [2]. Así pues, el motivo fundamental de este trabajo es establecer las bases para incorporar la experiencia adquirida en los sistemas programados en Ada al mundo de RT-CORBA.

El documento está organizado de la siguiente manera. El apartado 2 está dedicado a presentar las características básicas del middleware de distribución basado en RT-CORBA y el DSA de Ada y sus implementaciones. En el apartado 3 se hace un análisis detallado de los aspectos de planificación, mecanismos de distribución, y gestión de las llamadas remotas propuestos por los dos estándares y sus implementaciones. La evaluación y comparación de los tiempos de respuesta de llamadas a operaciones remotas para estas implementaciones se realiza en el apartado 4. Finalmente, el apartado 5 plantea las conclusiones y el trabajo futuro.

2. Middleware de distribución y tiempo real

En este apartado se van a describir los modelos de planificación de RT-CORBA y del DSA en la ejecución de peticiones remotas y cómo se puede soportar el modelo de transacción distribuida. Además se introducen brevemente las diferentes implementaciones que se van a analizar, todas ellas de código libre.

Las características principales de la arquitectura propuesta por RT-CORBA en su especificación [12] con respecto a la planificación son las siguientes:

- Uso de threads como entidades de planificación sobre los que se puede aplicar una prioridad RT-CORBA y que dispone de funciones de conversión a las prioridades nativas del sistema sobre el que se ejecute.
- Uso de dos modelos de propagación de la prioridad en llamadas remotas (siguiendo el modelo cliente-servidor): *Client_Propagated* (la invocación se ejecuta en el nodo remoto a la prioridad del cliente que viaja con el mensaje de la petición), y *Server_Declared* (todas las peticiones destinadas a un objeto se ejecutan a una prioridad prefijada en el servidor).
- En adición al punto anterior, permite realizar transformaciones de prioridad definidas por el usuario que modifican la prioridad asociada al servidor. Esto se hace con dos funciones llamadas *inbound* (transforma la prioridad antes de ejecutar el código del servidor) y *outbound* (transforma la prioridad con la que el servidor hace la llamada a otro servicio remoto).
- Define los grupos de threads (*Threadpools*) como mecanismo para gestionar las peticiones remotas. Considera que puede haber threads previamente creados o que se pueden crear dinámicamente, también puede haber varios grupos, o incluso que puede haber un máximo absoluto de threads por grupo.
- También define conexiones por bandas de prioridad. Este mecanismo se propone para reducir las inversiones de prioridad si se usa un protocolo de transporte que no usa prioridades.

El DSA de Ada por su parte no tiene previsto ningún mecanismo de transmisión de prioridades y por lo tanto su implementación se deja a criterio de la implementación. Lo que sí especifica es que se debe proveer de mecanismos para la ejecución de llamadas remotas concurrentes, así como también de mecanismos de espera a que la llamada remota retorne. La comunicación entre particiones activas se realiza de modo estándar utilizando el *Partition Communication Subsystem* (PCS). La concurrencia y los mecanismos de tiempo real están soportados por el propio lenguaje con tareas, tipos protegidos y el Anexo D. En [3] se propone un modo de realizar la transmisión de prioridades en el DSA, con un mecanismo que en principio es más potente que el de RT-CORBA porque permite la libre asignación de prioridades tanto en los procesadores como en las redes de comunicaciones que se usen.

La especificación de RT-CORBA también incorpora un capítulo dedicado a la planificación

dinámica, en la que básicamente se introducen dos conceptos:

- La posibilidad de introducir otras políticas de planificación además de la política de prioridades fijas, como por ejemplo, EDF (*Earliest Deadline First*), LLF (*Least Laxity First*), y MAU (*Maximize Accured Utility*). Se definen los parámetros de planificación como un contenedor que puede albergar más de un valor simple, y puede ser cambiado por la aplicación dinámicamente.
- El thread distribuido (*Distributable Thread*) que permite la planificación de principio-a-fin y la identificación de segmentos de planificación cada uno de los cuales se puede ejecutar en un procesador. Este concepto es similar a la transacción distribuida, define lo que llama puntos de planificación, y la forma de llevarlo a cabo se deja a la implementación.

El lenguaje Ada en su última revisión incluye en su anexo de sistemas de tiempo real (Anexo D) las políticas de planificación EDF y *Round Robin*. Sin embargo, en ningún caso contempla la existencia de la transacción distribuida.

Ni RT-CORBA ni el DSA de Ada contemplan la posibilidad de pasar parámetros de planificación a las redes de comunicaciones.

En este trabajo se analizan y evalúan dos implementaciones de RT-CORBA y otras dos del DSA:

- TAO [17] es una implementación de libre distribución de RT-CORBA que lleva varios años de evolución y desarrollo. Las aplicaciones se programan en C++ y la versión que hemos utilizado es la de Linux con TCP/IP. Se ofrece como una implementación de la especificación completa.
- PolyORB [14][19] se presenta como un middleware esquizofrénico capaz de soportar la distribución con diferentes personalidades como CORBA, RT-CORBA, DSA, o incluso RMI. Se distribuye con el compilador GNAT [1] y en principio está pensado para aplicaciones programadas en Ada. En la actualidad soporta CORBA y algunas nociones básicas de RT-CORBA (prioridades y su propagación). El DSA no está integrado y por tanto no es interoperable con RT-CORBA. La plataforma de ejecución es Linux y TCP/IP. En este trabajo, también hemos portado PolyORB al sistema operativo de tiempo real MaRTE OS [9] y le hemos dotado de la red de tiempo real RT-EP [8] (paso de testigo en anillo lógico sobre Ethernet estándar, con planificación por prioridades fijas).
- GLADE [13] es la implementación original de GNAT [1] del DSA de Ada para soportar el desarrollo de aplicaciones distribuidas con requisitos de tiempo real. Se sigue manteniendo hoy en la actualidad aunque evolucionará hacia PolyORB cuando esté completamente operativo. La planificación se realiza por prioridades fijas e implementa dos políticas para la distribución de prioridades al estilo de RT-CORBA (*Client Propagated* y *Server Declared*). De nuevo la plataforma de ejecución es Linux y TCP/IP
- RT-GLADE es una modificación de GLADE que optimiza sus características de tiempo real. Existen dos versiones: en la primera [6] se permite la asignación libre de prioridades en las llamadas remotas (tanto en los procesadores como en las redes de comunicaciones), y en la segunda [7] se propone una manera de incorporar al DSA la transacción distribuida y de soportar el uso de diferentes políticas de planificación en un sistema distribuido. La plataforma de ejecución es MaRTE OS [9] con la red RT-EP [8] (en este caso soporta dos tipos de planificación, una basada en prioridades fijas y la otra en contratos [2]).

3. Análisis de las implementaciones del middleware de distribución

El objetivo de este apartado es analizar los mecanismos de gestión de las llamadas remotas que utilizan las implementaciones de RT-CORBA o DSA para dar soporte a las especificaciones. También se discutirá si las soluciones adoptadas son deseables o no y si pueden ser mejorables, tanto para los estándares como para sus implementaciones.

3.1. Implementaciones de RT-CORBA y DSA

Desde el punto de vista de cómo se van a gestionar las llamadas remotas, TAO define varios elementos que se pueden configurar [15]:

- Número de ORBs. El ORB es la unidad de gestión de las llamadas a un servicio. Puede haber

varios o sólo uno, ya que cada ORB puede aceptar peticiones de diferentes puntos.

- La estrategia del servidor de concurrencia. Se definen dos modelos: el reactivo (*reactive*, en el que ejecuta un thread que va dando servicio a las múltiples conexiones), y un thread por conexión (*thread-per-connection*, para cada nueva conexión el ORB crea un thread que le da servicio).
- Los grupos de threads (*Threadpools*). Define dos tipos de grupos con dos comportamientos diferentes. En el modelo *ORB-per-Thread* cada thread tiene asociado un ORB que acepta y procesa los servicios solicitados. En el modelo *Leader/Followers* el usuario puede crear varios threads y el ORB va seleccionándolos alternativamente para que esperen y atiendan las nuevas peticiones.

Por otra parte, PolyORB define para la gestión de llamadas remotas los siguientes elementos configurables [14]:

- Políticas de los threads del ORB. Define cuatro políticas: sin threads (el ORB no crea threads y usa la tarea de entorno para procesar los trabajos), grupo de threads (define los que se crean en el arranque, el máximo que se mantiene en el grupo, y el máximo absoluto que puede contener el grupo), un thread por sesión (crea un thread por cada sesión que se abre), y un thread por requerimiento (crea un thread por cada requerimiento que llega y se destruye cuando el trabajo se completa).
- Configuración de las tareas del ORB. Básicamente se configuran los parámetros del grupo de threads.
- Políticas de control del ORB. Define cuatro políticas que afectan al comportamiento interno del middleware: sin threads (se hace un lazo en el que se monitorizan las operaciones de I/O y se procesan los trabajos), *Workers* (todos los threads son iguales y van monitorizando alternativamente las operaciones de I/O), *Half Sync/ Half Async* (un thread monitoriza las operaciones de I/O y las encola, y los otros las van procesando de esa cola), y *Leader/Followers* (similar a TAO, los threads se van turnando para esperar y procesar requerimientos).

La implementación del DSA realizada en GLADE define un grupo de threads para procesar los requerimientos con parámetros similares a los de PolyORB en cuanto al número de threads (mínimo en la creación, máximo estable y máximo absoluto), y utiliza otros dos threads intermedios para las peticiones; uno espera en la red la llegada de requerimientos, y el otro procesa esos requerimientos y elige a uno de los threads del grupo para que finalmente procese en trabajo.

Las modificaciones hechas en GLADE para obtener la primera versión de RT-GLADE eliminaban uno de los threads intermedios, de manera que había un thread esperando en la red y éste se encargaba de activar a uno de los threads del grupo para realizar el trabajo (a diferencia del *Half Sync/Half Async* de PolyORB sin cola intermedia). En la segunda versión, una API permite una configuración estática de los threads que van a ejecutar los trabajos, y que se quedan esperando directamente en la red. Esto se hace a través de la definición de puntos de conexión (*endpoints*) que además de las asociaciones con el thread remoto, soportan los parámetros de planificación. Estos parámetros, que pueden ser complejos, se asocian en el momento de la configuración y no tienen que viajar por la red cada vez que se requiere el servicio remoto.

Tanto TAO como PolyORB y GLADE utilizan las políticas de asignación de prioridades definidas en RT-CORBA. En cambio, en la primera versión de RT-GLADE se permite la asignación libre de prioridades a los servicios remotos y a los mensajes necesarios para la petición y respuesta. Esto permite el uso de técnicas de optimización de la asignación de prioridades en sistemas distribuidos.

Por otro lado, en la segunda versión de RT-GLADE [7], la definición de los puntos de conexión permite la programación de transacciones distribuidas sin más que especificar un identificador al principio de la transacción. Además, la transacción se ejecutará con los parámetros de planificación asociados en la configuración antes de arrancarla, tanto a los threads como a los mensajes que circulan por la red. Este concepto es similar al thread distribuido de RT-CORBA, salvo que esta especificación nunca tiene en cuenta la red. TAO implementa esta parte de planificación dinámica de RT-CORBA en la que se permite el cambio diná-

mico de los parámetros de planificación de un segmento de planificación [4].

3.2. Discusión

En este apartado se discuten las analogías y diferencias encontradas tanto en las implementaciones como en los propios estándares, y su adecuación para tiempo real, a través de los objetivos que debería perseguir el middleware de distribución de tiempo real. Entre estos objetivos estarían los siguientes:

- Permitir el análisis de planificabilidad de la aplicación completa. Aunque el middleware se ejecuta en el procesador y no es más que un usuario de las redes a través de interfaces claramente separadas, creemos que en muchos casos la redes se planifican conjuntamente con los procesadores [5], y por tanto, el middleware debería incorporar a través de los modelos adecuados los parámetros de planificación de la red. RT-GLADE puede servir como referencia [7].
- Transacciones o threads distribuidos. En la línea del punto anterior las transacciones o threads distribuidos deberían incorporar la información completa de planificación en procesadores y redes, ya sea en el modelo propuesto por RT-CORBA o en que se propone en RT-GLADE [7].
- Control de llamadas remotas. Las implementadas en TAO y PolyORB pueden servir de referencia, añadiendo un caso más en el que hay muchos threads, cada uno esperando siempre a la misma petición (al estilo de RT-GLADE en la segunda versión). Este caso puede ser útil en planificación flexible, cuando los threads ejecutan bajo contratos y el coste de enlazar y desenlazar contratos y threads puede ser muy alto. En el caso de que haya threads intermedios de gestión de las llamadas remotas (GLADE, RT-GLADE o PolyORB) es importante tener controlados sus parámetros de planificación. Este es el caso también de los grupos de threads en los que un thread puede ejecutar con unos parámetros diferentes cada vez.
- Permitir la libre asignación de parámetros de planificación. Esto se hace de modo genérico en RT-GLADE. En RT-CORBA hay solapamiento de conceptos entre la parte estática y la dinámica. Por ejemplo, si se sigue el concepto de thread distribuido parece que el cambio dinámico de parámetros de planificación puede solucionar el problema; en cambio, la parte estática impone unas normas en principio restrictivas respecto a la propagación de las prioridades.

Finalmente, y aunque éste pueda considerarse un criterio subjetivo, el middleware de distribución debería perseguir la facilidad de programación. En este aspecto, CORBA se aleja bastante de este punto.

4. Evaluación de algunas implementaciones

El objetivo de este apartado es dar una idea de la sobrecarga que introducen las implementaciones. Se va a utilizar una plataforma hardware que consta de dos procesadores AMD Duron a 800 MHz y una red Ethernet a 100 Mbps, y dos plataformas software como base:

- Linux con TCP/IP para evaluar las implementaciones de TAO, PolyORB y GLADE.
- MaRTE OS con RT-EP para evaluar PolyORB y RT-GLADE. Se utiliza la primera versión de RT-GLADE que es la que está operativa sobre la misma plataforma a la que se ha portado PolyORB.

Las pruebas van a consistir en la medición del tiempo de ejecución de una operación remota que suma dos enteros. La medida se realiza desde que se hace la llamada hasta que retorna el resultado. Esta operación se va a ejecutar de dos modos: sola y con otros cuatro clientes que realizan esta misma operación pero a una prioridad menor. El objetivo no es conseguir medidas exhaustivas de las plataformas, sino obtener una idea de las prestaciones que se pueden conseguir con el middleware. En todos los test la operación a evaluar se ejecuta 10.000 veces, y se evalúan los tiempos medio, máximo y mínimo.

En la Tabla 1 se muestran los resultados de las medidas realizadas sobre las plataformas Linux buscando las configuraciones del middleware que introducen menos sobrecarga. Para el caso de un solo cliente en TAO se ha utilizado el modelo de concurrencia reactivo con un único thread en el grupo. En PolyORB se ha usado el modelo sin threads para un cliente. Para el caso de cinco clien-

Tabla 1. Tiempos de ejecución en Linux para uno y cinco clientes

	Tiempos para un cliente (μs.)			Tiempos del cliente de mayor prioridad con cinco clientes (μs.)		
	Medio	**Máximo**	**Mínimo**	**Medio**	**Máximo**	**Mínimo**
TAO	914	8.044	861	1.072	8.237	846
PolyORB	1.276	3.176	1.214	7.610	12.675	4.447
GLADE	333	2.201	303	1.566	3.960	601
Sólo la red	113	366	109	--	--	--

Tabla 2. Tiempos de ejecución en MaRTE OS para uno y cinco clientes

	Tiempos para un cliente (μs.)			Tiempos del cliente de mayor prioridad con cinco clientes (μs.)		
	Medio	**Máximo**	**Mínimo**	**Medio**	**Máximo**	**Mínimo**
PolyORB	6.934	7.124	6.588	8.490	12.044	6.497
RT-GLADE	1.119	1.309	1.065	1.490	1.642	1.079
Sólo la red	727	735	723	--	--	--

tes tanto en TAO como en PolyORB se ha utilizado una configuración de un grupo de 5 threads con el modelo *Leader/Followers*. En GLADE se define un grupo estático de threads igual al número de clientes. El modelo de propagación de prioridades para TAO, PolyORB y GLADE ha sido el propagado por el cliente. Con objeto de relativizar las medidas de la sobrecarga del middleware se evalúa también el coste temporal del uso de la red. Así, la Tabla 1 incluye los tiempos (medio, máximo y mínimo) para el caso en el que se envía un mensaje y se espera la respuesta (el programa en el lado del servidor contesta nada más recibirlo).

En los resultados obtenidos para un cliente en Linux se puede observar que GLADE consigue mejores tiempos que TAO y que PolyORB, lo que demuestra que tiene un código más ligero. Son reseñables los tiempos máximos que se obtienen para TAO; anormalmente parecidos en los casos de uno y cinco clientes. La única explicación que cabe en este punto es que a pesar de que Linux no respete las prioridades, TAO sí que haga algún tipo de gestión de las mismas en los encolados de las peticiones tanto en el lado del cliente como en el del servidor. El aumento de los tiempos medios en GLADE y PolyORB para cinco clientes son los esperados cuando no hay gestión de prioridades fuera de la que ofrezca el sistema operativo.

En la Tabla 2 aparecen los resultados de las medidas realizadas sobre las dos plataformas MaRTE OS. La configuración de PolyORB es la misma que para Linux. La configuración del grupo de threads para RT-GLADE se hace igual al número de clientes, es decir, cinco. En cuanto a la red RT-EP se configura el parámetro correspondiente al retraso entre paquetes con un valor de 150 μs (este valor limita la sobrecarga en el procesador debida a la red). También se evalúa una transmisión simple en la red con idéntico fin que para el caso de Linux.

De los resultados obtenidos de la evaluación de la plataforma de tiempo real, se observa en primer lugar que el protocolo de red tiene una mayor latencia y hace que los tiempos de una transmisión simple de ida y vuelta sean mayores que en Linux; a cambio se obtiene una red predecible con una menor dispersión entre los valores de las medidas. Por otro lado, los tiempos mínimo y medio de RT-GLADE para un cliente también son mayores que los de GLADE sobre Linux, aunque el tiempo máximo se mantiene en una cota que indica una dispersión mucho menor. Una parte importante de los tiempos de respuesta obtenidos para RT-

GLADE se debe a la red, pero también al sistema operativo y al gestor de memoria dinámica que utiliza [10] (haciendo al conjunto predecible). Si nos fijamos en los tiempos de RT-GLADE para cinco clientes podemos ver que sólo el tiempo mínimo es peor que el de GLADE, aunque con una menor diferencia; en cambio el tiempo medio y sobre todo el máximo son ya ciertamente mejores. El aumento de todos los tiempos de RT-GLADE respecto al caso de un cliente es razonable y se puede justificar por los bloqueos que puede sufrir tanto en el procesador como en la red.

En la medida de los tiempos de PolyORB sobre MaRTE OS hemos encontrado una gran disparidad de las medidas para cinco clientes dependiendo de las prioridades utilizadas en los mismos. Después de analizar el código de PolyORB, encontramos que la implementación que hace del modelo *Leader/Followers* no se ajusta en realidad a dicho modelo. En lugar de tener un único thread esperando la petición remota para después ejecutarla y enviar la respuesta, sigue existiendo un thread que desacopla la recepción de mensajes de la red y la ejecución remota. Así, PolyORB implementa dos grupos de threads: uno para RT-CORBA (es necesario crearlo explícitamente para soportar el modelo de prioridades), y el otro que se corresponde con el soporte de la concurrencia de CORBA. Los threads que dan servicio a las peticiones de los clientes se toman del grupo de RT-CORBA, pero los thread intermediarios se toman del otro grupo y ejecutan a la prioridad intermedia del sistema, lo que puede introducir fuertes inversiones de prioridad dependiendo de las prioridades de los servidores. Esta es una parte que es preciso mejorar para garantizar cotas inferiores de los tiempos de respuesta de peor caso. De cualquier modo, las medidas reflejadas en la Tabla 2 para PolyORB con cinco clientes corresponden al menor tiempo máximo obtenido para la tarea de alta prioridad, y se consiguen con todos los clientes de baja prioridad ejecutando a una prioridad superior a la intermedia.

Así pues, respecto a los tiempos de PolyORB sobre MaRTE OS, de nuevo se demuestra, al comparar los resultados con los de RT-GLADE, que la implementación del DSA puede ser mucho más ligera que la de RT-CORBA. Comparando las pruebas de PolyORB para uno y cinco clientes se puede observar cómo existe una diferencia importante entre los tiempos mínimo y máximo para cinco clientes, lo que se debe a la inversión de prioridad introducida por las tareas intermediarias.

5. Conclusiones y trabajo futuro

En el trabajo presentado se ha realizado un análisis y evaluación de algunas implementaciones de middleware de distribución desde el punto de vista de su adecuación a la implementación de sistemas de tiempo real. En concreto se ha hecho énfasis en el modo en el que se realiza la gestión de las llamadas remotas, en los mecanismos de transmisión de los parámetros de planificación, y en la importancia de dar soporte a las transacciones o threads distribuidos.

En las medidas que se han realizado en Linux se ha podido comprobar cómo la implementación del DSA es más ligera que las implementaciones de RT-CORBA. Esto demuestra que Ada puede ser una buena opción para la programación de sistemas distribuidos, y podría encontrar un nicho en los sistemas empotrados distribuidos de tiempo real de tamaño medio. En las medidas realizadas sobre la plataforma de tiempo real basada en MaRTE OS se puede comprobar que todo va más despacio a cambio de tener predecibilidad.

A pesar de no ser la plataforma más eficiente, nuestro trabajo va a continuar con la experimentación sobre la plataforma PolyORB, por nuestra experiencia en el lenguaje Ada y en GLADE, y también porque pensamos que se puede mejorar. El objetivo es avanzar hacia la interoperabilidad en principio entre DSA y RT-CORBA, integrando el modelo de transacciones y nuevos mecanismos de planificación para procesadores y redes.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Educación y Ciencia del Gobierno de España dentro del proyecto THREAD (ref. TIC2005-08665-C03-02)

Referencias

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

[2] Aldea M., Bernat G., Broster I., Burns A., Dobrin R., Drake J.M., Fohler G., Gai P., González Harbour M., Guidi G., Gutiérrez J.J., Lennvall T., Lipari G., Martínez J.M., Medina J.L., Palencia J.C., Trimarchi M.. FSF: A Real-Time Scheduling Architecture Framework. Proc. of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2006, San Jose (CA, USA), 2006.

[3] Gutiérrez García J.J., y González Harbour M.. Prioritizing Remote Procedure Calls in Ada Distributed Systems. Proc. of the 9th International Real-Time Ada Workshop, ACM Ada Letters, XIX, 2, pp. 67–72, Junio 1999.

[4] Krishnamurthy Y., Pyarali I., Gill C., Mgeta L., Zhang Y., Torri S., Schmidt D.C.. The Design and Implementation of Real-Time CORBA 2.0: Dynamic Scheduling in TAO. Proc. of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04), Toronto (Canadá), Mayo 2004.

[5] Liu J.. Real-Time Systems. Prentice Hall, 2000.

[6] López Campos J., Gutiérrez J.J., y Michael González Harbour. The Chance for Ada to Support Distribution and Real Time in Embedded Systems. Proc. of the International Conference on Reliable Software Technologies, Palma de Mallorca, Spain, in LNCS, Vol. 3063, Springer, Junio 2004.

[7] López Campos J., Gutiérrez J.J., y Michael González Harbour. Interchangeable Scheduling Policies in Real-Time Middleware for Distribution. Proc. of the 11th International Conference on Reliable Software Technologies, Porto (Portugal), in LNCS, Vol. 4006, Springer, Junio 2006.

[8] Martínez J.M., y González Harbour M.. RT-EP: A Fixed-Priority Real Time Communication Protocol over Standard Ethernet. Proc. of the 10th International Conference on Reliable Software Technologies, York (UK), in LNCS, Vol. 3555, Springer, Junio 2005.

[9] MaRTE OS web page. http://marte.unican.es/

[10] Masmano M., Ripoll I., Crespo A., Real J.. TLSF: A New Dynamic Memory Allocator for Real-Time Systems. Proc of the 16th Euromicro Conference on Real-Time Systems, Catania (Italia), Junio 2004.

[11] Object Management Group. CORBA Core Specification. OMG Document, v3.0 formal/02-06-01, Julio 2003.

[12] Object Management Group. Realtime CORBA Specification. OMG Document, v1.2 formal/05-01-04, Enero 2005.

[13] Pautet L. y Tardieu S.. 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, Marzo 2000.

[14] PolyORB web page. http://polyorb.objectweb.org/

[15] Pyarali I., Spivak M., Schmidt D.C., y Cytron R.. Optimizing Thread-Pool Strategies for Real-Time CORBA. Proc. of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001), Snowbird, Utah, Junio 2001.

[16] Sun Developer Network, http://java.sun.com

[17] TAO web page. http://www.cs.wustl.edu/~schmidt/TAO.html

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

[19] Vergnaud T., Hugues J., Pautet L., and Kordon F.. PolyORB: a schizophrenic middleware to build versatile reliable distributed applications. Proc.of the 9th International Conference on Reliable Software Techologies, Palma de Mallorca (Spain), in LNCS, Vol. 3063, Junio 2004.