Programa Oficial de Postgrado en Ciencias, Tecnología y Computación
Máster en Computación
Facultad de Ciencias - Universidad de Cantabria

ADAPTACIÓN Y OPTIMIZACIÓN PARA UNA PLATAFORMA DISTRIBUIDA DE TIEMPO REAL DE UN MIDDLEWARE BASADO EN LOS ESTÁNDARES DE RT-CORBA Y ADA

Héctor Pérez Tijero

perezh@unican.es

Director:

J. Javier Gutiérrez García

Grupo de Computadores y Tiempo Real

Departamento de Electrónica y Computadores

Curso 2007 / 2008

Santander, Julio de 2008

Dedicatoria

A mi abuelo,

el primero que creyó en mí

PRÓLOGO

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) y dirigido y supervisado por D. J. Javier Gutiérrez García. También formaron parte fundamental en el desarrollo cada uno de los integrantes del grupo de *Computadores y Tiempo Real* de la Universidad de Cantabria, sin los cuales hubiera sido imposible finalizar este proyecto.

Parte del contenido de este trabajo ha sido publicado en:

> El Congreso Español de Informática (CEDI), II Simposio de Sistemas de Tiempo Real, Zaragoza, septiembre del 2007, bajo el título *CORBA & DSA: Análisis y evaluación de sus implementaciones desde la perspectiva de los sistemas de tiempo real.*
>
> El 13th International Conference on Reliable Software Technologies, Ada-Europe, Venecia (Italia), junio del 2008, con el título *Real-Time Distribution Middleware from the Ada Perspective.*

ÍNDICE

LISTADO DE FIGURAS Y TABLAS

1- INTRODUCCIÓN

Los sistemas distribuidos han sufrido un gran impulso en las últimas décadas como adaptación natural de la infraestructura informática de la empresa a su estructura física. Las compañías tienden a organizar y dividir el trabajo en varios departamentos, cada uno de los cuales puede llegar a estar incluso en distintos edificios pero ¿por qué no hacer lo mismo con las aplicaciones software? Esto es, utilizando otras palabras, ofrecer una serie de servicios sobre una red de ordenadores pero de forma transparente al usuario. En la actualidad esta visión está totalmente instaurada en nuestra vida cotidiana; cajeros automáticos, televisión por cable o servidores web son servicios que se utilizan prácticamente a diario. Este tipo de sistemas proporciona numerosas ventajas, entre las que se encuentran la descentralización, el reparto de funcionalidad o la tolerancia del sistema al fallo de un nodo. Sin embargo, existen otra serie de desventajas que deben tenerse en cuenta, relacionadas con el incremento de la complejidad en cuestiones de diseño, implementación, depuración y mantenimiento.

Podríamos hacer una clasificación previa de los tipos de sistemas distribuidos existentes según la capa del software en la que se basen. Así podemos tener:

- Sistemas operativos distribuidos: suelen ofrecer alta eficiencia y transparencia al usuario pero requieren el uso de un mismo sistema operativo subyacente.
- Middleware de distribución: se trata de una solución más flexible pero con menor eficiencia que la anterior que permite abstraer la distribución del software al usuario en sistemas heterogéneos.

Por otra parte, los sistemas de tiempo real, aquellos en los que su correcto funcionamiento no depende sólo de la corrección de los resultados sino que éstos se produzcan en un marco temporal correcto, están invadiendo cada vez cada nuestra vida cotidiana. Así, en la actualidad estos sistemas se utilizan en la industria (automovilística, aeroespacial, de telecomunicación, de electrodomésticos, etc.) y van incrementando ostensiblemente su complejidad, incorporando plataformas computarizadas distribuidas constituidas por decenas de nodos procesadores, que alojan diferentes aplicaciones independientes o acopladas entre sí, muchas de las cuales tienen requisitos de tiempo real e incluso niveles preestablecidos de calidad de servicio.

Este trabajo se centra en los sistemas distribuidos de tiempo real soportados por un middleware de distribución. En concreto se tratará de dotar al middleware de las capacidades necesarias para poder programar aplicaciones predecibles.

1.1- Antecedentes

Existen trabajos previos que de una forma u otra sirven de punto de partida para este proyecto. En primer lugar describiremos aquellos elaborados previamente en el propio grupo de *Computadores y Tiempo Real* de la Universidad de Cantabria, mientras que el último es un proyecto desarrollado originalmente en la *École Nationale Supérieure des Télécommunications* de Paris:

- MaRTE OS

  MaRTE OS [9] es un kernel de tiempo real para aplicaciones empotradas que sigue la norma *Minimal Real-Time POSIX.13* [24]. Este sistema operativo provee las interfaces POSIX para C y Ada.

- RT-EP (Real-Time ethernet Protocol)

  RT-EP [10] es un protocolo de paso de testigo para comunicaciones multipunto en aplicaciones de tiempo real. Está basado en el estándar Ethernet y puede por tanto utilizar el hardware basado en esta tecnología de uso tan común.

- RT-GLADE

  Es una modificación de GLADE [14], la implementación del anexo de sistemas distribuidos de Ada realizada por Adacore [1], que busca su adaptación para sistemas de tiempo real.

- PolyORB

  Nuevo middleware de distribución de Adacore que implementa los modelos de distribución de RT-CORBA y DSA

1.2- Sistemas distribuidos de tiempo real y modelado

En la actualidad existen multitud de ejemplos de sistemas de tiempo real en nuestra vida cotidiana. Por ejemplo, si pensamos en un coche, es posible que sin saberlo estemos utilizando varias decenas de procesadores encargados de controlar funciones tan diferentes como la radio, la inyección, el sistema de frenos, chequeos de niveles de líquidos o incluso la climatización del vehículo. La atención de algunas de estas funciones es prioritaria respecto a otras con las que comparte recursos. Por ejemplo, el control de los frenos ante una frenada de emergencia o el control de estabilidad ante un derrapaje debe tener prioridad sobre el hecho de que un pasajero esté cambiando la temperatura de su zona del habitáculo, además de tener que responder en un tiempo acotado. Así pues, el sistema no sólo debe tener la habilidad de realizar correctamente una acción, sino la necesidad de ejecutarla en un determinado espacio de tiempo (el plazo). Formalmente, un sistema de tiempo real queda caracterizado no sólo por su resultado lógico sino también por su respuesta temporal lo que lleva asociado un comportamiento *predecible* en la totalidad del sistema.

Los sistemas de tiempo real en los que no se puede permitir la pérdida de plazos se denominan *sistemas de tiempo real estricto*, y su programación debe asegurar un comportamiento predecible. El software del sistema de control de vuelo de un avión debe tener estas características, por ejemplo. Sin embargo, no todas las aplicaciones de tiempo real son críticas. Así, en una videoconferencia podríamos tolerar la pérdida de cierta calidad en la transmisión por el retraso en la respuesta temporal ante la llegada de nuevos datos a nuestro reproductor multimedia. Estos sistemas se denominan de *tiempo real no estricto*, pues se les permite incumplir algún plazo temporal y lo que sí deben garantizar es una cierta calidad del servicio. Para garantizar el comportamiento temporal del sistema se utilizan métodos de planificación de actividades junto con la abstracción y la representación de modelos del sistema sobre los que se pueden aplicar técnicas de análisis que predicen el comportamiento temporal del sistema.

Los mecanismos de planificación del sistema consisten en establecer las reglas que decidirán el orden de uso de un recurso (procesador o red en el caso de un sistema distribuido). La planificación utilizada por la mayoría de los sistemas se basa en prioridades fijas, pero existen otras estrategias de planificación basadas en plazos temporales o en contratos (reserva de recursos previamente negociados) [3] [6] [27].

La arquitectura típica de un sistema distribuido de tiempo real consta de varios procesadores interconectados a través de una o más redes de comunicaciones. El software del sistema se compone de varios threads concurrentes localizados en los procesadores que se intercambian mensajes a través de las redes de comunicaciones. También es posible que los threads de un mismo procesador compartan datos mediante los mecanismos de sincronización habitualmente utilizados en sistemas de memoria compartida. En los sistemas con asignación estática de threads y mensajes a procesadores y redes se plantean dos aspectos fundamentales: el análisis que determina la planificabilidad de los recursos (procesadores y redes de comunicación), es decir, si se van a cumplir o no todos los requisitos temporales incluso en el peor caso posible, y la asignación de prioridades a las acciones (tareas en los procesadores y mensajes en las redes de comunicación) que optimice la posibilidad de alcanzar todos esos requisitos temporales.

Un sistema distribuido de tiempo real se modela como un conjunto de *transacciones* cada una de las cuales se activa por la llegada de uno o más eventos externos, y representa un conjunto de actividades que serán ejecutadas por el sistema. Las actividades generan eventos que son internos a la transacción y que pueden activar a otras actividades de la transacción. Este modelo ha sido utilizado tradicionalmente en el análisis de tiempos de respuesta de las aplicaciones distribuidas de tiempo real [25].

1.3- Middleware de distribución para tiempo real

En esta sección introduciremos los principales paradigmas utilizados actualmente como middleware de distribución en sistemas de tiempo real, ofreciendo una visión general de cada uno de ellos y profundizando en aquellos que forman parte directamente del proyecto.

1.3.1- The Distributed Real-Time Specification for Java (DRTSJ)

El modelo de sistemas distribuidos para Java de tiempo real está siendo desarrollado por el *JSR-50 Expert Group* [21]. A falta de una especificación definitiva, en la actualidad sólo se puede realizar un esbozo de lo que serán sus bases para adaptar la especificación de tiempo real (RTSJ) a la de sistemas distribuidos (*Java Remote Invocation*). El objetivo es incorporar los conceptos de distribución y tiempo real a Java, y no adaptar el lenguaje para ofrecer este soporte. Existe gran semejanza entre el camino que está trazando esta especificación y la de RT-CORBA [13] que veremos posteriormente. Los puntos principales a destacar son:

- Dar soporte para diversas propiedades de la aplicación distribuida con requisitos de principio-a-fin.

  El soporte para estos requisitos (de naturaleza temporal o distinta) debe estar incluido en el middleware. Para ello, se introduce el concepto de *Distributable Thread* (utilizado también en RT-CORBA) que proporciona una abstracción del flujo de control de la aplicación distribuida entre nodos de principio-a-fin. Este concepto es similar al de la transacción distribuida.

- Permitir un marco (*framework*) de planificación fácilmente extensible.

  Se proporcionará la posibilidad de incorporar políticas de planificación a nivel de usuario (no sólo a nivel de la máquina virtual, *JVM*). Únicamente formará parte del estándar la planificación basada en prioridades fijas, siendo el resto opcionales.

- No considerar el uso de protocolos de red de tiempo real por defecto.

  Permitiendo así la interoperabilidad entre sistemas de tiempo real y aquellos que no lo son.

Dado que todavía esta especificación no está completa y quedan aspectos por solventar, en este trabajo se ha descartado un análisis más profundo y se centrará en los otros dos estándares, remitiendo este capítulo a trabajos futuros.

1.3.2- CORBA y RT-CORBA

CORBA [12] es un modelo de distribución de objetos que sigue el paradigma cliente-servidor. Su núcleo es el ORB (*Object Request Broker*), el cual posibilita la comunicación entre los nodos cliente y servidor. Sus principales características son:

- Independencia del lenguaje utilizado (multilanguaje).

  La clave está en el lenguaje descriptivo IDL en el que se definen los objetos CORBA. En la actualidad existen especificaciones para el mapeado de los tipos de datos a los lenguajes Ada, Java y C.

- Independencia de la plataforma (interoperable).

  El protocolo de transporte genérico (GIOP) permite la comunicación entre distintos ORBs.

La comunicación entre nodos se realiza a través de distintas entidades de la arquitectura CORBA. En el caso más sencillo, las siguientes entidades constituirían el entorno necesario para llevar a cabo dicha comunicación:

- Object Request Broker

  El ORB ofrece los mecanismos necesarios para comunicar de forma transparente los nodos cliente con los objetos del nodo servidor, haciendo que las invocaciones remotas tengan la apariencia de simples llamadas locales, abstrayendo por tanto la localización del objeto y la forma de comunicarse con él.

- Adaptador de objetos

  El adaptador de objetos se puede entender como una capa externa que envuelve al objeto remoto CORBA caracterizándole mediante distintas propiedades. La necesidad de incluir una capa intermedia responde a la necesidad de conseguir un ORB lo menos complejo posible, además de permitir la caracterización comentada. Al principio se utilizó el BOA (*Basic Object Adapter*) que básicamente se correspondía con un mapa de objetos activos en el sistema. Sin embargo, su simplicidad hizo que muchas empresas añadieran nuevas características que acabaron por minar la portabilidad entre ORBs de distintos vendedores. Por ello, se decidió desarrollar el actual POA (*Portable Object Adapter*) que incluye distintos parámetros de configuración cómo políticas de atención o procesamiento de llamadas remotas, activación de los objetos, etc.

- GIOP

  Este protocolo se sitúa por encima del nivel de transporte de la pila *OSI* [26]. GIOP proporciona la interoperabilidad entre ORBs a nivel de protocolo y se compone principalmente de tres partes:

  - Common Data Representation (*CDR*): Aquí se establecen las normas para mapear los datos a enviar por la red a partir del IDL, entre los que se encuentran el alineamiento o el formato de almacenamiento de los datos.
  - Formato del mensaje: Gestiona quién debe tratar y qué deben hacer cada uno de los ocho tipos de mensajes disponibles.
  - Transporte del mensaje: Modo de adaptar GIOP a la capa de transporte a utilizar (TCP, UDP u otros).

- Interoperable Object Reference

  Esta referencia identifica unívocamente al objeto remoto. Hablando de forma amplia, podemos definir este parámetro como el contenedor de la información de contacto del objeto CORBA. Aquí se incluyen los detalles de todos los protocolos y puntos de escucha que el ORB puede utilizar para atender las peticiones entrantes.

La extensión de la especificación CORBA para su uso en entornos con requisitos de tiempo real se denomina RT-CORBA [13]. Esta extensión incorpora nuevas interfaces tales como RT-POA, RT-ORB, mapeados de prioridad o políticas de planificación que permiten su utilización tanto para sistemas no críticos (agencias de viaje o carrito de compras on-line) como críticos (sistemas de control). En concreto, las nuevas interfaces incorporadas son:

- RT-ORB

  Es una extensión del ORB al que le añade funciones de creación/eliminación de entidades específicas de tiempo real (mutexes, thread pools, o políticas de planificación de tiempo real) y permite la asignación de prioridades de los threads dedicados a tareas internas del ORB.

- RT-POA

  Extensión del POA que ofrece soporte para políticas que definen el esquema de transmisión de prioridades, gestión de llamadas remotas, atención a peticiones de conexión o selección/configuración de protocolos disponibles.

- Priority y Priority Mapping

  Define una prioridad genérica (independiente del sistema operativo subyacente), y mapea las prioridades nativas sobre prioridades RT-CORBA (rango 0- 32767). Este mapeado no está estandarizado.

- Mutex

  API de acceso a los mutexes implementados por el ORB, proporcionando recursos para la sincronización del acceso a los recursos del sistema.

- RTCurrent

  Interfaz para manejar las prioridades del thread activo en el sistema.

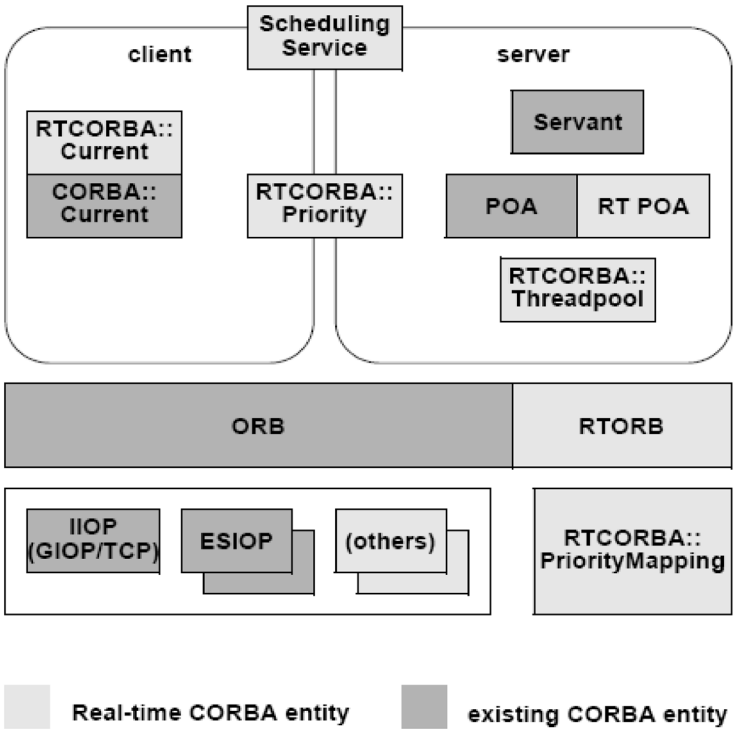

Figura 1 Extensión RT-CORBA [13]

La Figura 1 muestra un esquema de la arquitectura RT-CORBA en la que se pueden distinguir y diferenciar los servicios propios de CORBA y de RT-CORBA.

1.3.3- Anexo de Sistemas Distribuidos de Ada (DSA)

La característica principal del Anexo de Sistemas Distribuidos del lenguaje Ada (DSA, Distributed Systems Annex) es la posibilidad de permitir al usuario final desarrollar sus aplicación siguiendo el mismo procedimiento independientemente de si va a ejecutarse en un entorno distribuido o no.

Para el lenguaje Ada un sistema distribuido se define como la interconexión entre uno o más nodos procesadores y opcionalmente cierta cantidad de nodos de almacenamiento. También se 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 (*Remote Call Interface*) y objetos distribuidos (*Remote Types*). En el DSA 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 thread de control propio.

El DSA del lenguaje Ada no está expresamente diseñado para soportar aplicaciones de tiempo real, y muchos de los aspectos que afectan al comportamiento de tiempo real se dejan dependientes de su implementación. Sin embargo, existen trabajos previos que demuestran que es posible utilizarlo para tales propósitos [7][8]. El DSA permite programar sistemas distribuidos de una forma sencilla, y puede incorporar en el middleware de tiempo real conceptos como la transacción distribuida [8] y la planificación flexible [2].

1.4- Objetivos

Como objetivo general se marca el desarrollo de un middleware de distribución para sistemas empotrados de tiempo real que permita el uso de diversos paradigmas de distribución y aúne las mejores cualidades de cada uno bajo una misma aplicación. Debe incluir soporte para distintas políticas de planificación, ofreciendo una interfaz genérica a las aplicaciones. Además, deberá ser capaz de manejar no sólo parámetros de planificación sencillos como prioridades fijas, sino también otros de mayor complejidad como los basados en contratos, e incorporar los mecanismos necesarios para dar soporte al modelo transaccional.

Este trabajo establece las bases para conseguir dicho objetivo final a través de los siguientes puntos:

- Análisis de los modelos de distribución de los estándares actuales desde el punto de vista de su aplicación en sistemas de tiempo real, con el propósito de identificar sus posibles carencias.
- Análisis de las implementaciones de libre distribución de dichos modelos, haciendo especial énfasis en la gestión interna de las llamadas remotas y en las políticas de planificación disponibles.
- Mejora de las características de tiempo real de PolyORB, el middleware con el que vamos a trabajar, mediante la portabilidad a una plataforma de tiempo real y la incorporación de los mecanismos necesarios para optimizar su comportamiento en dicho escenario.
- Evaluación de las soluciones incorporadas al middleware de distribución en una plataforma de tiempo real.

2- ANÁLISIS DE LOS ASPECTOS DE TIEMPO REAL DE LOS ESTÁNDARES DE DISTRIBUCIÓN

El objetivo de esta sección es analizar los aspectos concernientes a la planificación dentro de los dos modelos de distribución que hemos introducido en el capítulo 1 y que nos proponemos abordar: RT-CORBA y el DSA de Ada. Nótese que en este capítulo hablamos de estándares y no de implementaciones, tema que será abordado en capítulos posteriores. Introduciremos asimismo la terminología básica utilizada en los sistemas de distribución.

2.1- Terminología de los sistemas de distribución

En un entorno distribuido la comunicación entre nodos se realiza de forma transparente al usuario mediante la incorporación de los mecanismos necesarios para establecer la comunicación. De esta forma, durante el transcurso de una llamada remota intervienen dos entidades:

- Stub: Es el código que redirige la llamada, sustituyendo el cuerpo de la operación remota invocada.
- Skeleton: Es el código encargado de ejecutar la operación en el nodo remoto.

Además, los parámetros de la invocación se transmiten utilizando una representación de streams, dando lugar a dos conceptos:

- Marshalling: Es la conversión a stream.
- Unmarshalling: Es la recuperación de los parámetros a partir del stream.

2.2- RT-CORBA

El estándar RT-CORBA clasifica los sistemas distribuidos basándose en la naturaleza del sistema a utilizar:

- Estáticos: En estos sistemas la carga del sistema es conocida de antemano, permitiendo la realización de un análisis de planificabilidad a priori. El sistema de planificación utilizado por este tipo de sistemas se basa en prioridades fijas. El análisis previo nos permite mapear los requisitos temporales de la aplicación a prioridades.
- Dinámicos: En este caso, la carga del sistema no está preestablecida y por tanto puede ir variando con el tiempo por lo que una planificación basada en prioridades fijas puede resultar no ser la más adecuada para satisfacer los requisitos de tiempo real de nuestro sistema en dichos entornos.

Las características principales de la arquitectura propuesta por RT-CORBA en su especificación [13] con respecto a la **planificación estática** 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.

En el capítulo dedicado a la **planificación dinámica** 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*), o 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 mediante la identificación de segmentos y puntos de planificación. Los segmentos de planificación representan partes de código asociadas a un conjunto determinado de parámetros de planificación establecidos específicamente por la aplicación. Los puntos de planificación definen los instantes en los que se pueden cambiar los parámetros de planificación de un segmento que forma parte de un thread distribuido (estos puntos se definen en la Tabla 1). El concepto de thread distribuido es similar a la transacción distribuida.

Finalmente, hagamos notar que RT-CORBA no contempla la posibilidad de pasar parámetros de planificación a las redes de comunicaciones.

Tabla 1 Correspondencia de los puntos de planificación en RT-CORBA

Número	Instante de la petición
1	Al crearse un nuevo thread hijo o al comenzar un segmento marcado como planificable (BSS)
2	Al actualizar un segmento planificable (USS)
3	Al finalizar la ejecución de un segmento planificable (ESS)
4	Al realizar el envío de una petición
5	Al recibir una petición
6	Al realizar el envío de una respuesta
7	Al recibir una respuesta

2.3- DSA

El anexo de sistemas distribuidos del lenguaje Ada no está diseñado para soportar aplicaciones de tiempo real aunque existen trabajos previos que demuestran que es posible utilizarlo para tales propósitos [7][8].

El DSA de Ada no tiene previsto ningún mecanismo de transmisión de prioridades y por lo tanto su esquema de ejecución 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), que tiene una interfaz definida por el lenguaje de manera que la implementación del PCS pueda ser independiente del compilador y del sistema de ejecución.

La concurrencia y los mecanismos de tiempo real están soportados por el propio lenguaje con tareas, tipos protegidos y el anexo de tiempo real (Anexo D). En [14] se propone un modo de realizar la transmisión de prioridades en el DSA siguiendo el mismo esquema que en RT-CORBA. Asimismo, en [8] se propone un mecanismo que permite optimizar el comportamiento de tiempo real mediante la libre asignación de prioridades tanto en los procesadores como en las redes de comunicaciones que se usen.

El lenguaje Ada en su última revisión [19] incluye en su anexo de sistemas de tiempo real las políticas de planificación *EDF* y *Round Robin*. Sin embargo, en ningún caso contempla la existencia de la transacción distribuida ni tampoco la posibilidad de pasar parámetros de planificación a las redes de comunicaciones.

3- ANÁLISIS DE LAS IMPLEMENTACIONES

En este capítulo realizaremos un análisis de las implementaciones objeto de estudio de este trabajo. Se hará énfasis en los mecanismos utilizados para la gestión de llamadas remotas y el acceso a los parámetros de planificación.

3.1- GLADE

3.1.1- Introducción

GLADE [14] es la implementación original de GNAT [1] del DSA de Ada para soportar el desarrollo de aplicaciones distribuidas con requisitos de tiempo real. En la actualidad se sigue manteniendo aunque no de forma activa sino con funciones de soporte técnico exclusivamente. Su futuro reside en evolucionar hacia la personalidad DSA de PolyORB cuando esté completamente operativo. Las plataformas de ejecución del software son Linux como sistema operativo y redes basadas en IP. GLADE se divide en dos partes bien diferenciadas [14]:

- 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.

3.1.2- Análisis de la gestión interna del Middleware

El comportamiento interno del middleware no es configurable pues sigue un único patrón. En este caso, la implementación del DSA realizada en GLADE define un grupo de threads para procesar los requerimientos con ciertos parámetros configurables relacionados con el número de threads existentes en el sistema (similares a los utilizados en PolyORB):

- Task pool minimum size: Este valor define el número de threads creados en tiempo de compilación.

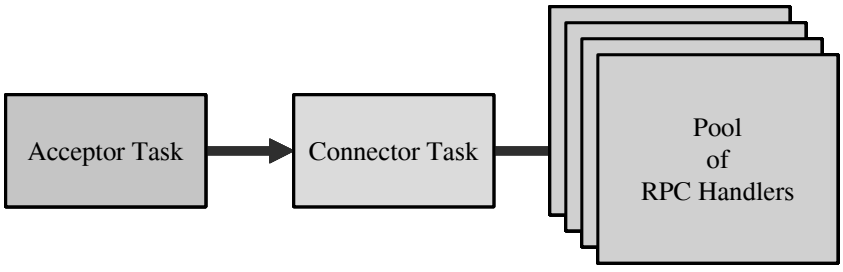

Figura 2 Gestión interna de peticiones remotas en GLADE

- Task pool high size: Es el máximo número de threads que pueden coexistir en el sistema, contando los creados en el inicio y los creados dinámicamente. Una vez que un thread ha terminado su trabajo, se comprueba que no se sobrepasa este límite, eliminándose si así fuera.
- Task pool maximum size: Límite absoluto de threads. Una vez sobrepasado, no se crearán más threads dinámicamente y las peticiones pasarán a encolarse.

Previamente al procesado de la llamada remota, se utilizan otros dos threads intermedios para las peticiones entrantes; uno espera en la red la llegada de eventos externos (*Acceptor Task*), y el otro procesa esos requerimientos (*Connector Task*) y elige a uno de los threads del grupo de threads para que finalmente procese en trabajo (*RPC Handlers*), tal y como se ilustra en la Figura 2.

3.1.3- Planificación

La planificación se realiza por prioridades fijas e implementa dos políticas para la distribución de prioridades que siguen el mismo esquema que RT-CORBA (*Client Propagated* y *Server Declared*).

Como se observa en la Figura 3, parte de la petición remota se ejecutará a una prioridad no controlada por el usuario (en este caso, la máxima prioridad del sistema). Además, no se permite la asignación libre de prioridades ni se tiene en cuenta a la red como entidad planificable. Debido a estas limitaciones, se inició un proyecto en el grupo de Computadores y Tiempo Real de la Universidad de Cantabria para adaptar GLADE a sistemas de tiempo real: RT-GLADE [7].

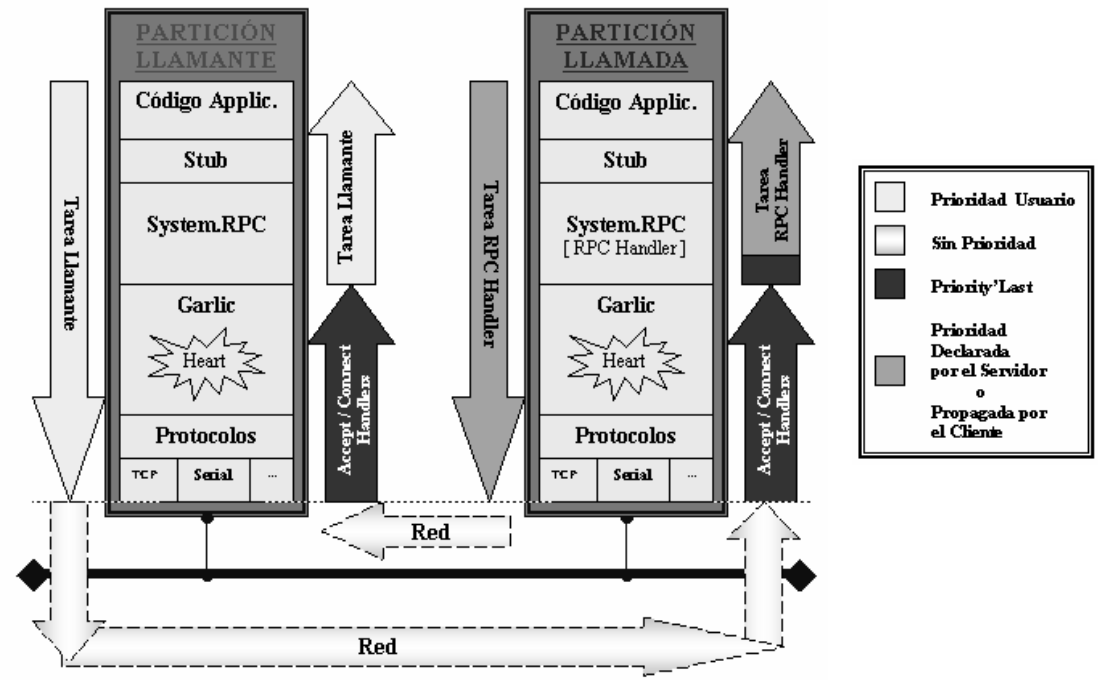

Figura 3 Esquema de prioridades durante una llamada remota en GLADE [28]

3.2- RT-GLADE

3.2.1- Introducción

RT-GLADE [7] es una modificación de GLADE que optimiza sus características de tiempo real y solventa las limitaciones anteriormente descritas en cuanto a asignación de prioridades y gestión interna de llamadas remotas. Para ello, en un proyecto previo se portó este middleware a un sistema operativo de tiempo real, MaRTE OS, y se le incorporaron dos redes de tiempo real: RT-EP y CAN. Existen dos versiones de RT-GLADE con dos modelos diferentes de gestión de llamadas remotas y configuración del sistema.

3.2.2- Análisis de la gestión interna del Middleware

Las modificaciones hechas en GLADE para obtener la primera versión de RT-GLADE eliminaban uno de los threads intermedios (*Connector Task*), 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. Este patrón de comportamiento se ilustra en la Figura 4.

En la segunda versión de RT-GLADE [8], 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. Además, la definición de los puntos de conexión permite la programación de transacciones distribuidas sin más que especificar un identificador al principio de la transacción (ver Figura 5). 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 última especificación nunca tiene en cuenta la red.

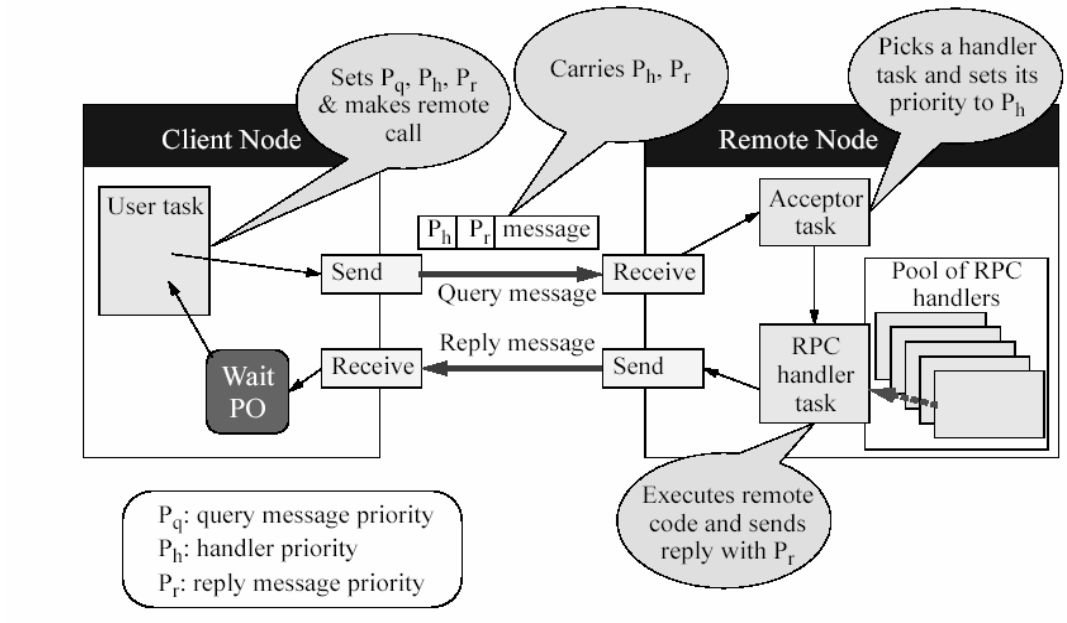

Figura 4 Gestión de prioridades en la primera versión de RT-GLADE [8]

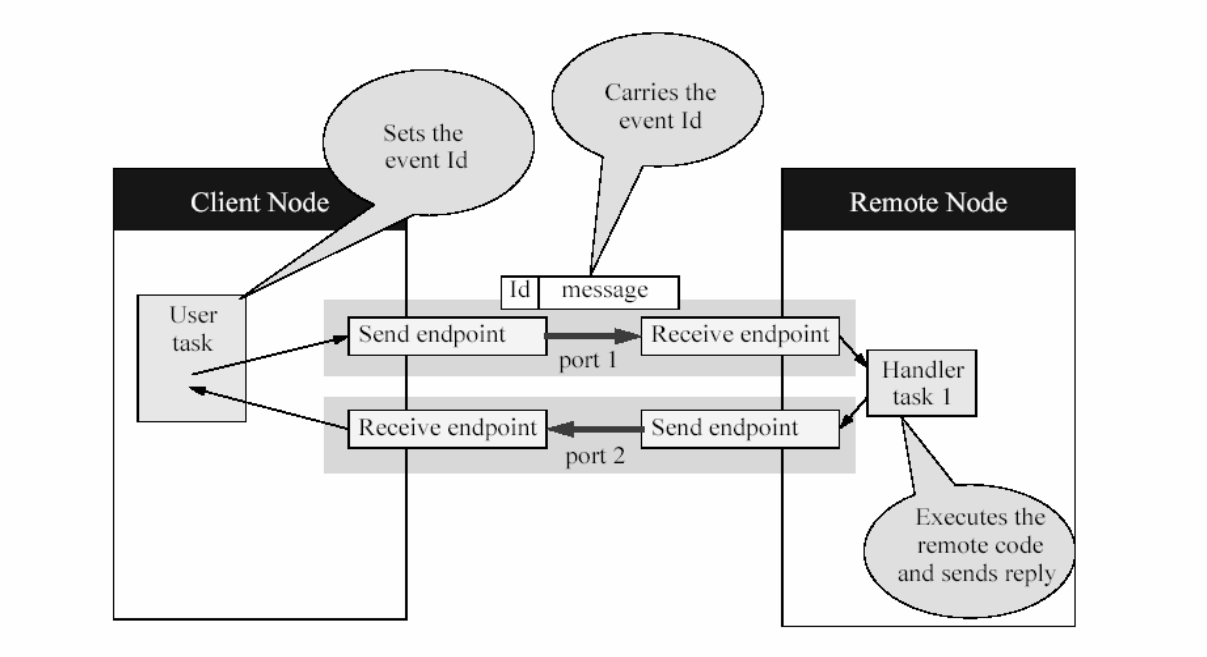

Figura 5 Patrón de gestión interna basado en endpoints [8]

3.2.3- Planificación

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, es decir, se consideran tanto los procesadores como las redes de comunicaciones como entidades planificables. Esto permite el uso de técnicas de optimización de la asignación de prioridades en sistemas distribuidos. Como se observa en la Figura 4, tanto la prioridad del objeto remoto como la del mensaje de respuesta son los parámetros que se transmiten a través de la red.

En la segunda versión [8] 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. Concretamente, se implementan dos políticas de planificación basadas en prioridades fijas y en contratos FSF [2]. Como ya apuntamos antes, los parámetros de planificación se asocian a los *endpoints*. 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, tal y como se muestra en la Figura 5. En este caso, sólo el parámetro que identifica el evento es enviado junto con la petición remota y el nodo destino será el encargado de recuperar los parámetros correspondientes (*table-driven*).

3.3- PolyORB

3.3.1- Introducción

PolyORB [15] es un middleware de distribución escrito íntegramente en lenguaje Ada. Su característica principal reside en ofrecer soporte para distintos paradigmas de distribución como

CORBA o DSA, permitiendo la interacción entre los distintos modelos bajo la misma plataforma software. Se distribuye con el compilador GNAT [1] y en principio está pensado para aplicaciones programadas en Ada. En la actualidad soporta CORBA, algunas nociones básicas de RT-CORBA (prioridades y su propagación) y DSA. Las plataformas de ejecución del software son Linux, Windows o Solaris como sistemas operativos y redes basadas en IP.

La arquitectura de PolyORB se divide en tres capas bien diferenciadas entre sí como se representa en la Figura 6: la capa de aplicación (denominada *personalidad de aplicación*), la capa neutra o núcleo de distribución y la capa de protocolo (denominada *personalidad de protocolo*). Este tipo de arquitectura permite combinar las distintas personalidades, ya sean referentes a los modelos de distribución o a los protocolos de red, en una misma aplicación permitiendo por tanto la interoperabilidad y la integración bajo una misma plataforma de los distintos paradigmas disponibles.

3.3.2- Personalidades de aplicación y protocolo

La personalidad de protocolo es la encargada de gestionar el mapeado de las peticiones en mensajes de red. Actualmente existen dos personalidades implementadas:

- SOAP: Protocolo basado en XML para servicios web
- GIOP: Protocolo genérico de transporte para objetos CORBA. Dispone de cuatro perfiles distintos
  - IIOP: Basado en TCP/IP (orientado a conexión)
  - SSIOP: Utilización de sockets SSL
  - DIOP: Basado en UDP/IP (no orientado a conexión)
  - MIOP: Utilización de la comunicación IP multicast

La personalidad de aplicación es la capa de adaptación entre la aplicación y el middleware, proporcionando los mecanismos necesarios para la comunicación entre ambas entidades. Actualmente existen cuatro personalidades:

- CORBA (Common Object Request Broker) y RT-CORBA: PolyORB soporta parcialmente este modelo estándar de distribución definido por la OMG e incluye algunos servicios CORBA como el servicio de nombres o el de eventos.
- DSA (Distribution System Annex): Es el anexo de distribución del lenguaje Ada.

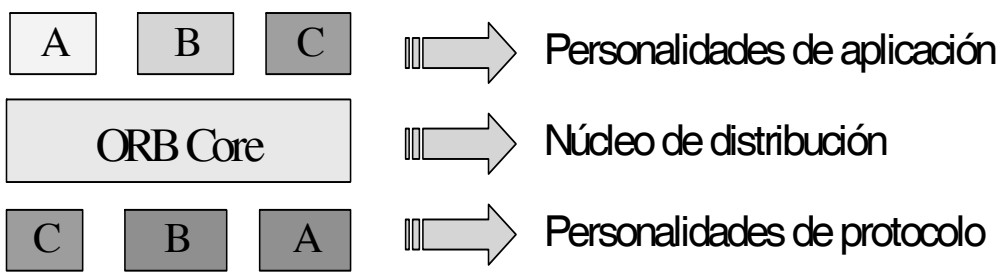

Figura 6 Arquitectura de PolyORB

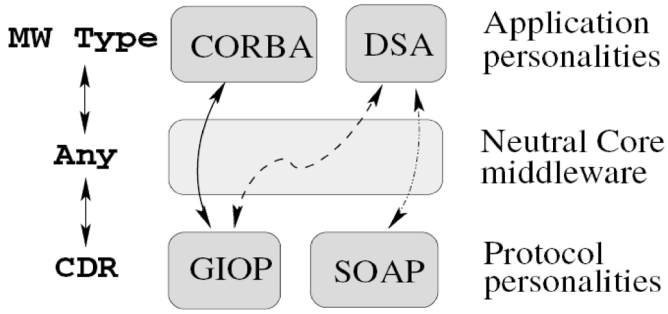

Figura 7 Interacción entre personalidades en PolyORB [22]

- AWS (Ada Web Server): Sigue el modelo de distribución basado en servicios web. Permite interactuar directamente con peticiones HTTP a través del protocolo SOAP.
- MOMA (Message Oriented Middleware For Ada): Implementa la distribución basada en paso de mensajes.

Las dos primeras son las únicas disponibles a través de la última versión hecha pública, mientras que las dos restantes se encuentran todavía en desarrollo.

3.3.3- Interoperabilidad entre personalidades

Como hemos apuntado ya, la característica principal de este middleware es la interoperabilidad entre diferentes modelos de distribución. PolyORB es conocido como el middleware esquizofrénico ya que su arquitectura separa la parte de aplicación de la parte de red, ofreciendo por tanto dos tipos de personalidades. La interacción entre ambos tipos de personalidades se realiza a través del núcleo del middleware mediante una conversión de los tipos de datos de cada personalidad a un tipo intermedio denominado *Any*. Este proceso se ilustra en la Figura 7.

De esta forma, conseguimos que, como se observa en la Figura 8, una personalidad de aplicación cualquiera, dígase AP3, pueda comunicarse con otra, denominada AP1, a través del protocolo PP3.

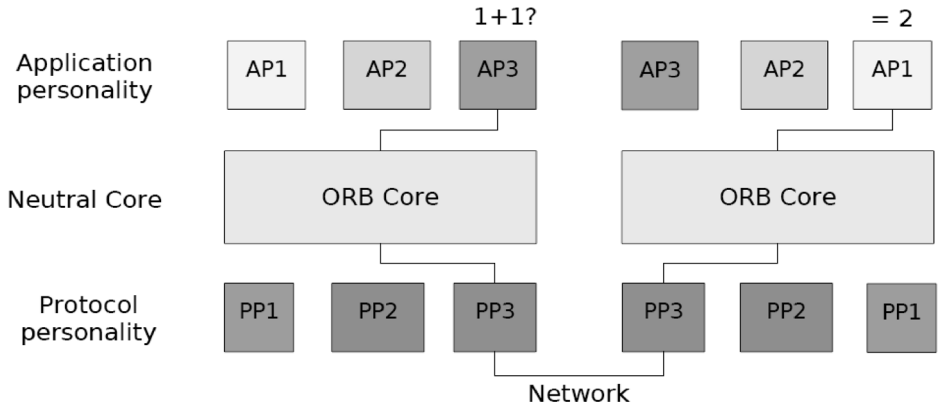

Figura 8 Interoperabilidad entre personalidades en PolyORB

3.3.4- Análisis de la gestión interna del Middleware

PolyORB se presenta como un middleware altamente configurable, permitiendo adaptar la forma de interactuar de las personalidades de aplicación y protocolo y el ORB (ver Figura 9) según diversas políticas de gestión y configuración de threads:

- **Configuración de los threads del ORB**. Básicamente define los perfiles y las restricciones a utilizar dentro del grupo de threads del sistema. Existen tres perfiles:
  - *No Tasking:* La aplicación completa será monothread, esto es, sólo existirá un thread en el sistema para la ejecución de todas las operaciones (ejecución secuencial).
  - *Full Tasking:* Este perfil activa todas las capacidades de gestión y sincronización de threads siendo, por consiguiente, el sistema multithread (ejecución concurrente).
  - *Ravenscar*: Esta configuración activa las capacidades de gestión y sincronización de threads según las restricciones del perfil de *Ravenscar* [19]. Este perfil es utilizado en sistemas de tiempo real de alta integridad.
- **Políticas de los threads del ORB**. Representan las políticas que se deben seguir en la creación de threads para atender las llamadas remotas o gestionar los trabajos internos. Están estrechamente relacionadas con la semántica de los mensajes intercambiados entre nodos. Define cuatro políticas:
  - *No_Tasking (sin threads)*: el ORB no crea threads y usa el thread de entorno para procesar los trabajos.
  - *Thread_per_Request (un thread por requerimiento)*: Los threads se crean dinámicamente según vayan llegando nuevas peticiones Los threads sólo se crearán al llegar una mensaje del tipo *Request*, es decir, el servidor utilizará sus propios threads del sistema cuando esté gestionando la conexión y/o cualquier otro tipo de mensajes. El thread se eliminará del sistema una vez completada la petición.
  - *Thread_per_Session (un thread por sesión*): Se creará un thread dinámicamente por cada petición de conexión que se realice. Al igual que en el caso anterior, durante el procesado de otro tipo de mensajes el thread principal será la encargada de realizar el trabajo y sólo le cederá el puesto una vez alcanzada la petición en sí (*Handle Request Execution*). El thread permanecerá en el sistema hasta que la conexión sea cerrada**.**

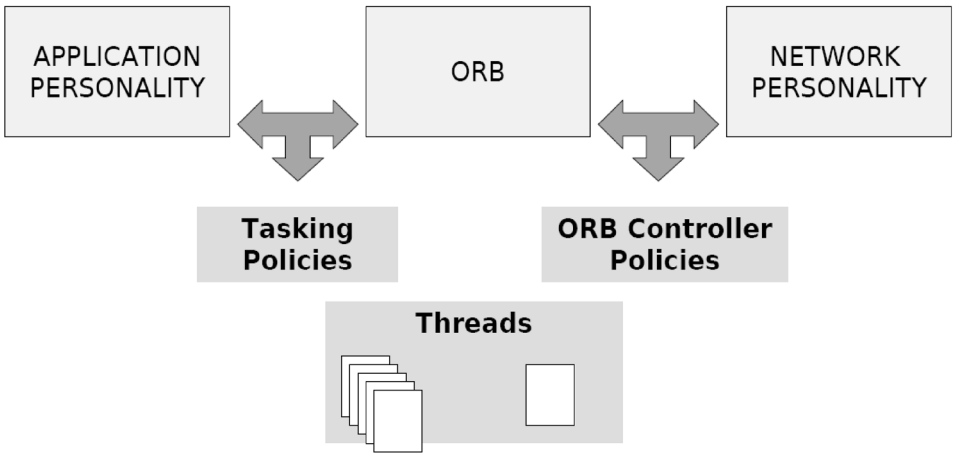

Figura 9 Gestión interna de PolyORB

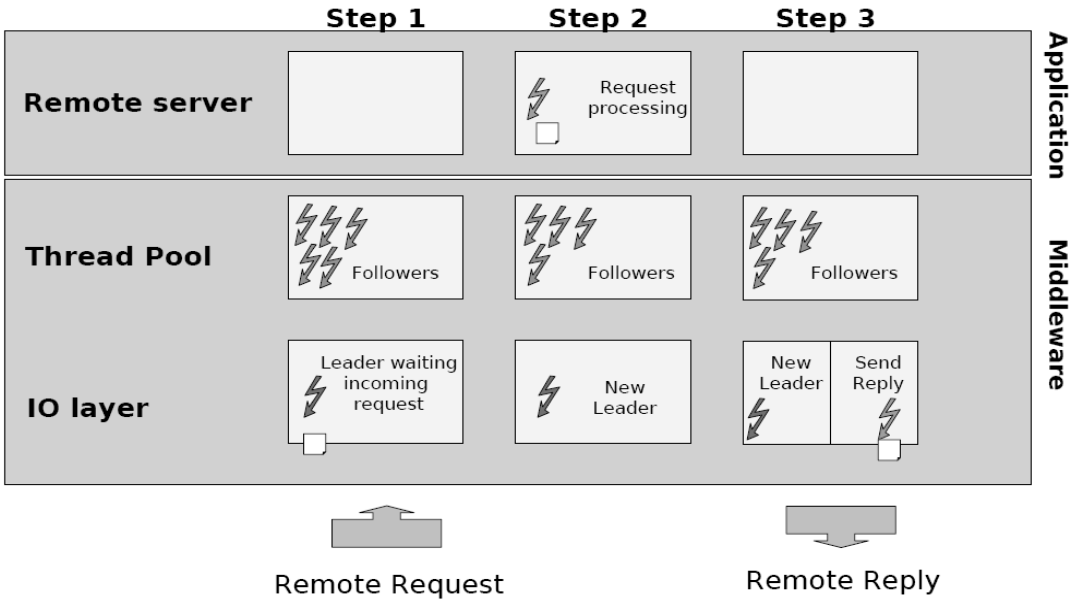

Figura 10 Patrón Leader / Followers

- *Thread_Pool (grupo de threads*): los trabajos se realizaran mediante el pool de threads creado al inicio. En el caso de que existiera un mayor número de peticiones que de threads disponibles, éstas se encolarían hasta poder ser procesadas. Existen tres parámetros en el archivo de configuración para esta política:

  - *min_spare_threads*: Este valor define el número de threads creados en tiempo de compilación.

  - *max_spare_threads*: Es el máximo número de threads que pueden coexistir en el sistema, contando los creados en el inicio y los creados dinámicamente. Una vez que un thread ha terminado su trabajo, se comprueba que no se sobrepasa este límite, eliminándose si así fuera.

  - *max_threads*: Límite absoluto de threads. Una vez sobrepasado, no se crear más threads dinámicamente y pasan a encolarse las peticiones.

- **Políticas de control del ORB**. Aquí se engloban todas las políticas destinadas a controlar el modo de operar del ORB (*main loop*): gestión del ORB, asignación de threads internos y procesado de I/O. Define cuatro políticas que afectan al comportamiento interno del middleware:

  - *No_Tasking:* 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 y el procesado de mensajes. Este patrón es de propósito general.

  - *Leader/Followers:* Un thread del pool, denominado *leader*, es el encargado de monitorizar las operaciones de I/O tal y como se observa en la primera etapa de la Figura 10. Al llegar un evento externo, este thread será el encargado de promocionar un nuevo *leader* (segunda fase), que pasará a monitorear la capa de IO, y de procesar el requerimiento de principio a fin (esto es, el thread a la espera de eventos externos será el encargado de procesar toda la petición y enviar la respuesta (tercer y último

paso)). Esta política favorece la atención de gran número de peticiones de poca carga computacional.

- *Half Sync/Half Async:* Esta política separa los threads de alto nivel de los de bajo nivel. De esta forma, un thread monitoriza las operaciones de I/O y las encola, mientras que el resto del threads del pool van procesando los mensajes de esa cola. Este patrón es indicado para sistemas con gran carga computacional en la ejecución (se busca reducir el promedio de los tiempos de ejecución).

Estas políticas no son ortogonales entre sí por lo que no todas las combinaciones son posibles. En algunos casos, como en el HS / HA o el L&F, sólo la política *Thread_Pool* permite el comportamiento deseado.

PolyORB no distingue entre threads de aplicación y aquellos destinados a realizar acciones del ORB. Así pues, dentro del pool que hemos creado internamente, se añadirán asimismo los threads de aplicación, pudiendo ir éstos a interactuar con el nivel de red por otra petición distinta a la suya. Este comportamiento se conoce como *nested upcall* e intenta maximizar la eficiencia de uso de todos los threads en el sistema y evitar ciertas situaciones de bloqueo en sistemas monothread, por ejemplo, en el caso en el que el cliente realiza una petición de *callback* remota. Sin embargo, este modelo de actuación no es muy indicado para sistemas de tiempo real, dónde priman los tiempos de respuesta y no debería permitirse la posibilidad de que una petición quede pendiente de entrega a la espera de que el thread de aplicación termine sus tareas adicionales.

3.3.5- Planificación

PolyORB sigue el mismo esquema de planificación que la parte estática de RT-CORBA, basada en prioridades fijas y cuya propagación se realiza mediante las políticas *Client Propagated* o *Server Declared*. Esta arquitectura es común tanto en la personalidad CORBA como en la del DSA.

Existen trabajos previos que indican que es posible mejorar el comportamiento de tiempo real de una aplicación si ésta es capaz de especificar libremente las prioridades. Por ello, se propone como uno de los objetivos trasladar las modificaciones establecidas en [8] a PolyORB, permitiendo incluir la red como entidad planificable y dotándole de una nueva política de gestión de llamadas remotas.

4- ADAPTACIÓN DE POLYORB A UNA PLATAFORMA DE TIEMPO REAL

PolyORB es un middleware de distribución orientado al uso en sistemas de tiempo real ya que ofrece soporte para el estándar RT-CORBA. Sin embargo, no implementa de forma completa la especificación, limitándose exclusivamente a una parte del capítulo dedicado a la planificación estática. La personalidad del DSA, en una versión publicada durante la redacción de esta memoria, aún cuando el estándar no ofrece soporte para sistemas de tiempo real, sigue el esquema de planificación estática basado en prioridades fijas de RT-CORBA. Sin embargo, la versión utilizada en este trabajo no permitía utilizar ninguna política para especificar parámetros de planificación (por ejemplo, prioridades) por lo que fue modificada para completar el estudio.

Por otro lado, el soporte incluido en el software original es insuficiente para optimizar las aplicaciones de tiempo real, lo que ha motivado la incorporación de una serie de mejoras y extensiones cuyo objetivo es dotar al conjunto de mayor predecibilidad:

- Soporte para un nuevo sistema operativo
- Nueva personalidad de protocolo
- Nuevas políticas de control y gestión del ORB

4.1- Soporte para un sistema operativo de tiempo real

PolyORB se ha portado a un sistema operativo, MaRTE OS, que ofrece soporte a la arquitectura de prioridades implementada en el middleware. Los principales cambios realizados se detallan a continuación:

- Eliminación de todas aquellas llamadas no soportadas en MaRTE OS. Dentro de este apartado, podemos encontrar los paquetes relacionados con los *sockets* o aquellas funciones que requerían del uso de un sistema persistente de ficheros.
- Incorporación a la librería **`libmgnat`** de MaRTE OS de algunos ficheros de la distribución de GNAT que se necesitaban para la correcta compilación de la aplicación distribuida.
- Creación de nuevos *scripts* de compilación de MaRTE OS que hacen la instalación de PolyORB independiente de cualquier otra aplicación, además de incorporar soporte para el nuevo estilo de compilación basado en proyectos GPS.
- Sustitución en el compilador **`po_gnatdist`** de todas las llamadas que deberían ejecutarse con las llamadas proporcionadas por MaRTE OS.

4.2- Nuevas políticas de control y gestión del ORB

El desarrollo de nuevas políticas de control y gestión del ORB nos permiten disponer de un control absoluto sobre todas los threads y mensajes utilizados en la aplicación minimizando de esta forma las inversiones de prioridad. Dos políticas han sido añadidas al conjunto ofrecido por PolyORB: ReAdy To gO (RATO) y Thread Per Target (TPT). El comportamiento que otorgan ambas al conjunto del sistema es similar al descrito previamente para RT-GLADE en su segunda versión [8].

RATO es una política que controla el comportamiento interno del ORB. Concretamente, se ocupa de la gestión del lazo principal del ORB, de los eventos de I/O y del procesado de las peticiones remotas. Por tanto, en resumidas cuentas, determina qué acción ejecutará el thread seleccionado. Esta política provee un control completo sobre los threads del middleware, siendo su principal característica el poder procesar una petición remota de principio a fin, esto es, sin incurrir en ningún cambio de contexto distinto a los marcados por la propia prioridad de la transacción. Este procesamiento de llamadas remotas se basa en la asociación del par thread - *endpoint*, mediante el cual un thread queda asociado a los parámetros de planificación establecidos en el *endpoint*.

Bajo la política *Thread Per Target* los threads son creados exclusivamente en tiempo de inicialización. Pasado ese instante, ningún thread será creado dinámicamente en el sistema. Podemos considerar esta política como un pool de threads estático y no anónimo, pues permite la realización de asociaciones thread - *endpoint* y la atención directa de las peticiones entrantes por parte de esta dupla.

Ambas políticas, TPT y RATO, están concebidas para ser utilizadas conjuntamente y así minimizar los cambios de contexto y permitir la asociación thread - *endpoint*.

La transacción distribuida está relacionada con este modelo basado en *endpoints* a través de un parámetro, denominado *event Id*, que es el único que se transmite por la red. De esta forma, se evita el envío de estructuras de datos que en ciertos casos pueden llegar a ser complejas [2][3], siendo ineficiente su transmisión. El nodo cliente debe especificar simplemente el evento de la transacción que va a llevar a cabo, siendo este identificador el que permita en el nodo remoto recuperar los parámetros de planificación previamente configurados en tiempo de compilación. Este comportamiento se reflejó en la Figura 5 del capítulo 3.

4.3- Nueva personalidad de protocolo

Las personalidades de protocolo adaptan los eventos ocurridos en la red a mensajes propios del middleware. En nuestro caso, hemos adaptado el protocolo de tiempo real RT-EP a PolyORB para añadir la red como un elemento planificable más de nuestro sistema.

Este protocolo de red ha sido implementado bajo el protocolo de transporte genérico de CORBA, denominado GIOP. Esta capa de transporte nos asegura la interoperabilidad con cualquier otro ORB que implemente RT-EP, además de proveer una capa de fragmentación de mensajes en el nivel de transporte. Esta personalidad ha sido desarrollada de dos formas distintas: en la primera los parámetros de planificación viajan por la red, denominada RTC-RTEP, y en la segunda los parámetros se asocian estáticamente a los *endpoints* (de manera similar a la segunda versión de RT-GLADE [8]) y se denomina RATO-RTEP.

4.3.1- RTC-RTEP

Esta personalidad ha sido desarrollada sólo con propósitos de prueba. Puede ser utilizada con cualquiera de las políticas de control y creación de threads originales de PolyORB, incorporando además la posibilidad de especificar los parámetros de planificación (en este caso prioridades) de los mensajes en la red (tanto la petición como la respuesta). Estas prioridades son establecidas a través de un API en el lado del cliente:

```
procedure Configure_RTEP_Parameters (IOR_Ref : in String;
                                     P_in    : in RTEP_MAC.Priority;
                                     O_out   : in RTEP_MAC.Priority;
                                     P_out   : in RTEP_MAC.Priority;
                                     Event_ID: in Natural);
```

Configure_RTEP_Parameters: Función que establece los parámetros de planificación asociados a un evento tanto en el nodo local como en el remoto

IOR_Ref : Información de contacto con el objeto remoto. El string debe tener el formato: IOR:XXX
P_in: Prioridad de envío de la petición en la red
O_out: Prioridad de ejecución del objeto remoto
P_out: Prioridad de la respuesta en la red
Event_ID: Identificador de evento

A la hora de incorporar los parámetros de planificación de la red en el estándar CORBA se han manejado dos elementos claves: el *IOR* y el *Service Context*. El primero de ellos nos habilita la posibilidad de informar a los nodos cliente que vamos a utilizar dicha característica durante la comunicación. Según se puede observar en la Figura 11, el IOR se compone de una lista de perfiles soportados, cada uno con su identificador exclusivo. En nuestro caso, el perfil RT-EP simplemente almacenará la versión utilizada, la localización del objeto remoto (estación y puerto de escucha), el identificador dentro del nodo remoto (*Object Key*) y el campo *Components*, que es la estructura que podemos extender para incluir información adicional como, en nuestro caso, que se va a utilizar el modelo de prioridades de la RT-EP.

Por otra parte, la información contextual de cada invocación es transportada de forma oculta a través del campo *Service Context*. Aquí se puede especificar cualquier propiedad inherente a la petición, desde una prioridad hasta la codificación utilizada. En nuestro caso, este campo será el encargado de transportar los parámetros de planificación relativos a la red y a la ejecución en el nodo remoto de forma encapsulada en un mensaje estándar GIOP tal y como se ve en la Figura 12.

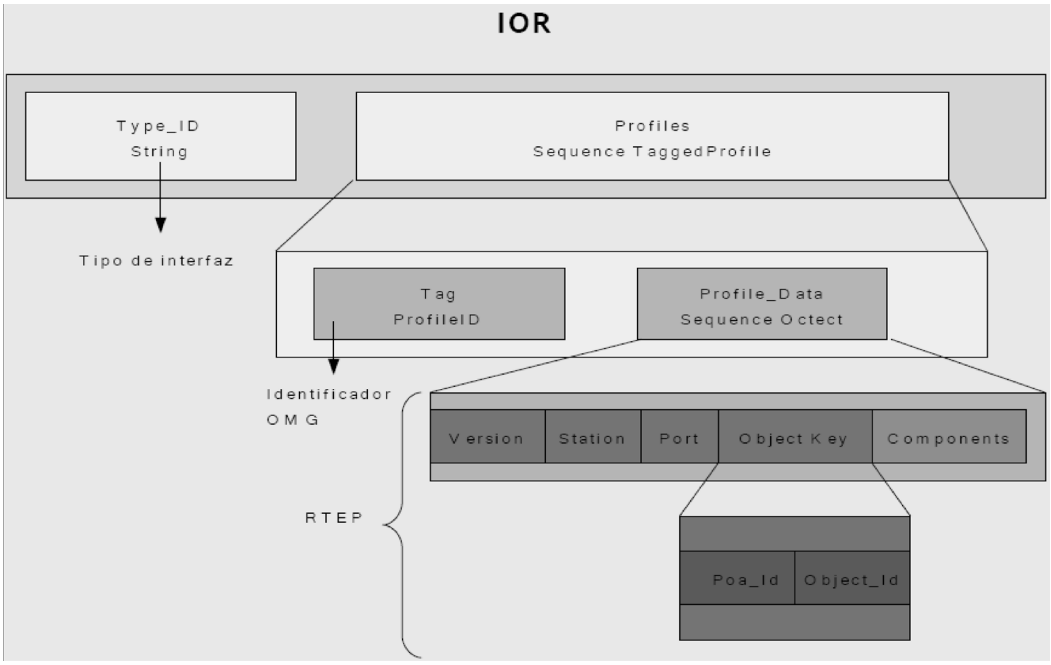

Figura 11 Estructura del Interoperable

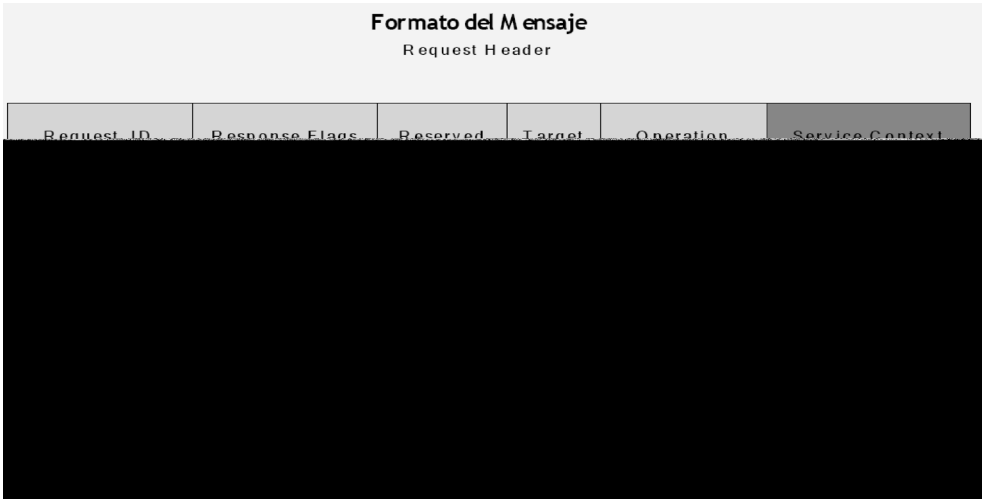

Figura 12 Formato de mensaje tipo *request* en GIOP

Una vez establecidos los parámetros de planificación tanto en la red, a través del proceso descrito anteriormente, como del objeto remoto, mediante la utilización de las políticas establecidas por RT-CORBA, nos ocuparemos del manejo de estos parámetros por parte del middleware. PolyORB, al utilizar cualquiera de las políticas de control preestablecidas, presenta una falta de gestión de dichas prioridades entre los niveles de atención a los eventos externos y el procesado previo de mensajes por parte del ORB. Cómo se representa en la Figura 13, un cliente realiza una petición remota mediante el envío de un mensaje a la red con prioridad *X*, estableciendo como prioridades de la operación remoto y del mensaje de respuesta *Y* y *Z* respectivamente. En el lado servidor, el thread que atiende la llegada de peticiones presenta una prioridad que no puede ser establecida explícitamente por parte de la aplicación, siendo ésta la prioridad por defecto del sistema y la que se utilizará para atender cada una de las peticiones entrantes. Durante este intervalo de tiempo, la aplicación es propensa a sufrir inversiones de prioridad. Este comportamiento incrementa los tiempos de respuesta de los requerimientos más prioritarios, limitando de esta forma la planificabilidad del sistema.

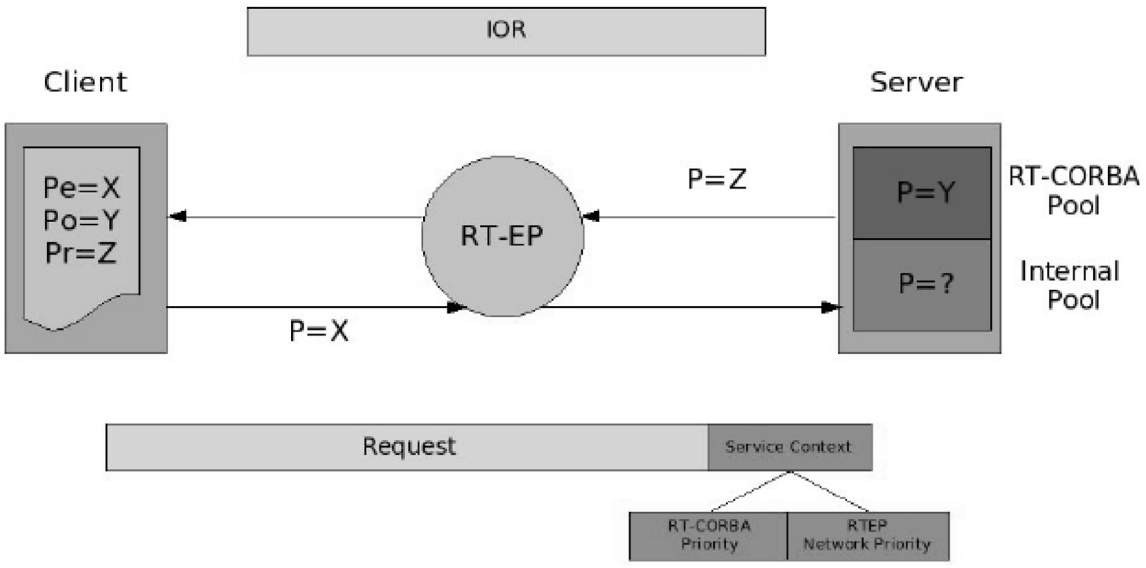

Figura 13 Diagrama de funcionamiento de RTC-RTEP

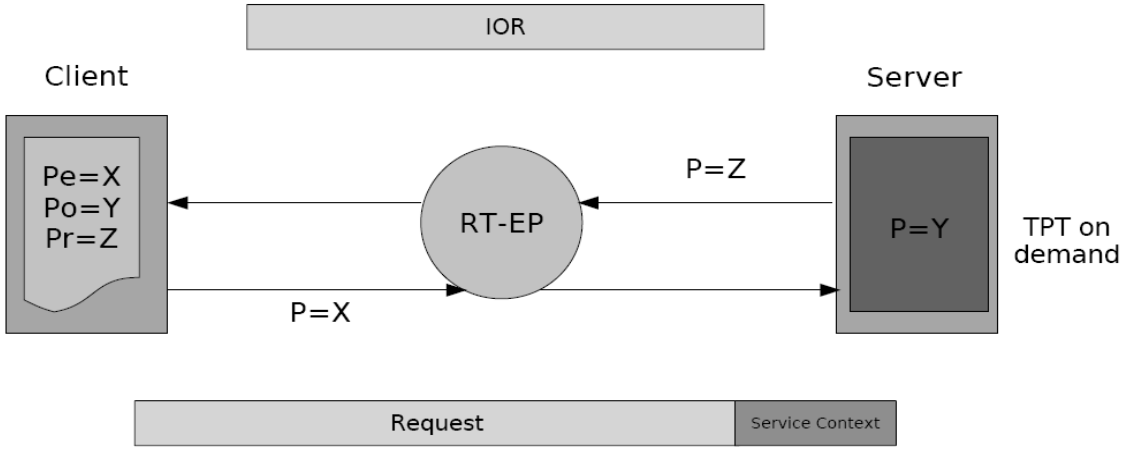

Figura 14 Diagrama de funcionamiento de RATO-RTEP

4.3.2- RATO-RTEP

Para solventar los problemas descritos en la sección anterior, se ha diseñado una nueva versión de la personalidad del protocolo. Dicha personalidad aprovecha las características añadidas por las políticas RATO y TPT en la gestión de los mensajes de red y el empleo de los parámetros de planificación de forma totalmente controlada (ver Figura 14). Estos parámetros son definidos para la transacción en tiempo de configuración pero, a diferencia de RTC-RTEP, no son enviados a través de la red. Este esquema de funcionamiento no utiliza el entorno RT-CORBA ya que los requisitos de tiempo real son gestionados independientemente de forma directa, esto es, caracterizando nuestro sistema completamente a nivel de transacción. Esa metodología resulta adecuada en sistemas distribuidos con requisitos de tiempo real de tamaño medio (decenas de nodos) o cuando la planificación no se basa en una simple prioridad sino en el uso de parámetros más complejos, como los contratos; en este caso, el envío de éstos resulta ineficiente desde el punto de vista de la red aparte del coste de negociar y cancelar contratos que puede llegar a ser muy alto.

Al igual que en el caso de RTC-RTEP, se muestra un diagrama simplificado de su funcionamiento interno, ilustrando las prioridades en cada parte de la llamada remota.

La configuración de parámetros se realiza a través de la API que se detalla a continuación. Ambos nodos serán configurados en tiempo de inicialización especificando tanto los parámetros de planificación cómo el identificador de evento y el puerto de escucha en la red.

```
procedure Create_Receive_Endpoint
     (Params    : in Message_Scheduling_Parameters_Ref;
      Dest_Node : in RTEP_MAC.Station_ID;
      Event_ID  : in Natural;
      Channel   : in RTEP_MAC.Channel;
      Endpoint  : out RTEP_MAC.Endpoint_Id);
```

Create_Receive_Endpoint: Función que crea una tarea Ada y la asocia a un nuevo endpoint junto con los parámetros de planificación definidos

Params: Parametros de planificación genéricos. Debe ser formateados a través de la función *Set_Scheduling_Params*
Dest_Node: Estación de respuesta asociada a este endpoint.
Event_ID: Identificador de evento
Channel: Puerto de escucha de llamadas remotas
Endpoint: Identificador de endpoint

```
procedure Create_Send_Endpoint
     (Param     : in Message_Scheduling_Parameters_Ref;
      Dest_Node : in RTEP_MAC.Station_ID;
      Event_ID  : in Natural;
      Channel   : in RTEP_MAC.Channel;
      Endpoint  : out RTEP_MAC.Endpoint_Id);
```

Create_Send_Endpoint: Asocia la tarea Ada actual a un nuevo endpoint junto con los parámetros de planificación definidos

Params: Parametros de planificación genéricos. Debe ser formateados a través de la función *Set_Scheduling_Params*
Dest_Node: Estación destino de la petición asociada a este endpoint
Event_ID: Identificador de evento
Channel: Puerto de espera para la respuesta a la invocación remota
Endpoint: Identificador de endpoint

```
procedure Set_Scheduling_Params
     (From_Params   : in Message_Scheduling_Parameters;
      To_Params     : in out Message_Scheduling_Parameters_Ref);
```

Set_Scheduling_Params: Formatea el tipo de datos de planificación a un formato genérico

From_Params: Parámetros de planificación a los que realizar la conversión
To_Params: Parámetros de planificación convertidos

Finalmente, se muestra una tabla a modo de resumen con las diferencias más notables entre ambas personalidades:

Tabla 2 Diferencias entre personalidades de la RT-EP: RTC y RATO

	RTC-RTEP	**RATO-RTEP**
Prioridades	A través de *Service Context*	Configuradas previamente
Endpoints(servidor)	Uno por objeto [1]	Uno por canal a utilizar
Endpoints(cliente)	Uno por objeto	Uno por canal a utilizar
Parámetros de planificación	Asociamos los datos al identificador de petición (Request_ID)	Asociado al endpoint
Threads	Según política	Dedicados por cada canal a utlizar
Nivel I / O	Monothread [2]	Multithread
Múltiples objetos (lado servidor)	Sólo un punto de escucha en el sistema que servirá a todos los objetos	Creamos los puntos de escucha bajo demanda (no están asociados al objeto)
Sesiones GIOP (Buffers)	Uno	Uno por cada canal a utilizar
Canales de rx	Monocanal	Multicanal bajo demanda
Inversión de prioridad (Peor caso)	La petición llega al servidor justo después de haber recibido otro evento. Hasta que no se añada de nuevo la fuente de eventos no se mandará otro thread a supervisar el I/O	La petición llega al servidor justo después de otra petición sobre el mismo canal[3]. Se espera hasta que la petición sea servida
Cambios de contexto para el thread de mayor prioridad	Según política	Ninguno

1. Actualmente, en caso de existir un número mayor de objetos todos utilizarían el mismo punto de escucha. Esto se debe a que sólo se define un punto de escucha internamente y, por tanto, todos los objetos serán invocados a través de éste.
2. El sistema viene configurado para utilizar un sólo monitor de eventos externos y, por tanto, sólo un thread podrá monitorizar cada vez todas las fuentes de eventos externos asociadas
3. Esta opción es eliminada con una configuración apropiada en tiempo de diseño

4.4- Adaptación de la personalidad DSA

La versión PolyORB 2.3, utilizada en este proyecto, no presenta una personalidad del anexo de sistemas distribuidos de Ada lo suficientemente desarrollado como para emplearse en sistemas de tiempo real. La reciente versión 2.4 sí que incluye soporte para el uso de una planificación basada en prioridades fijas.

El compilador *po_gnatdist* ha sido modificado para utilizar los nuevos scripts de compilación de MaRTE que hemos desarrollado para esta personalidad. Además, se han tenido que suprimir las dependencias de los paquetes no soportados por MaRTE (por ejemplo, los *sockets*) y añadir los ficheros propios del anexo E (*s-parint* y *s-dsaser*) al *runtime* de MaRTE.

Un requisito de esta personalidad es la necesidad de un servidor de nombres para registrar u obtener la dirección del objeto remoto. Para ello hemos tenido que portar este servicio a la plataforma de trabajo y configurar nuestra aplicación de modo que pueda acceder al nodo del servidor de manera transparente. En el apéndice se incluye una descripción más detallada de este servicio y su uso. Otro inconveniente es la imposibilidad de configurar las políticas de control del ORB ya que esta personalidad asume una política del tipo *Workers* por defecto. Para poder configurar el sistema con la política RATO + TPT, se ha modificado el código de la herramienta de compilación y así poder incluir esa posibilidad. De esta forma, la especificación de los parámetros de planificación para la personalidad DSA se hará a través de la misma API descrita anteriormente para el perfil de protocolo RATO-RTEP.

A diferencia de RT-CORBA, PolyORB-DSA se encuentra en un estado experimental por lo que la adaptación a la plataforma de tiempo real no se puede considerar como finalizada.

4.5- Arquitectura de middlewares y protocolos de tiempo real

A modo de resumen, se muestra en la Figura 15 un compendio de los middlewares de distribución que soporta nuestra plataforma de tiempo real, con las políticas de control y planificación disponibles, así como los protocolos de red.

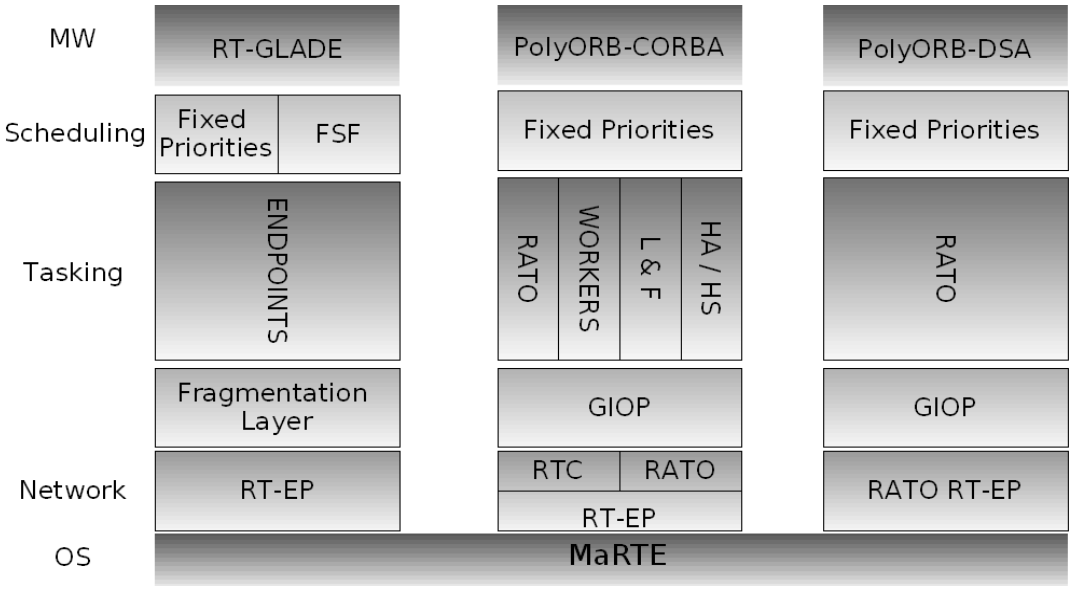

Figura 15 Sistemas de distribución disponibles en nuestra plataforma de tiempo real

5- EVALUACIÓN

El objetivo de este apartado es estimar la sobrecarga que introducen las implementaciones de middleware de distribución sobre las que se ha trabajado. 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:

- Una plataforma de propósito general basada en Linux y un kernel versión 2.6.10 junto con una red TCP/IP para evaluar las implementaciones de PolyORB (versión 2.3) y GLADE (versión 2007). El objetivo es contrastar los resultados de las implementaciones de partida, que se ofrecen sobre esta plataforma, con las modificaciones propuestas sobre la plataforma de tiempo real.
- Una plataforma de tiempo real basada en MaRTE OS, versión 1.7, con una red RT-EP para evaluar las versiones adaptadas de PolyORB versión 2.3 (PolyORB-CORBA, PolyORB-CORBA-RATO y PolyORB-DSA) y RT-GLADE (adaptado a partir de GLADE 2007).

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. De esta forma, para el primer caso conseguiremos una estimación de la sobrecarga introducida por el middleware, mientras que para el segundo se evalúa su comportamiento en un entorno con requisitos de tiempo real. 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 tests la operación a evaluar se ejecuta 10.000 veces, y se evalúan los tiempos medio, máximo y mínimo junto con la desviación estándar y el porcentaje de muestras que se incluyen en el intervalo entre el máximo y el 10% de la diferencia entre el máximo y el valor medio.

En aquellos entornos que utilizan planificación flexible, los threads se ejecutan bajo contratos siendo el coste asociado a su creación y renegociación muy alto para el sistema. Para minimizar los cambios de contexto y satisfacer estos requisitos, se ha presentado en este trabajo una política de control que sigue un patrón basado en *endpoints*, donde threads dedicados esperan por la llegada de eventos externos y posteriormente procesan los eventos recibidos de principio a fin. El patrón *Leader/Followers* sigue un concepto similar pero sin utilizar threads dedicados y por ello será la política elegida en la evaluación de las prestaciones de la implementación original de PolyORB.

La Tabla 3 y la Tabla 4 muestran los resultados de las medidas tomadas en la plataforma Linux. Para el caso de un cliente, en ambas implementaciones se ha utilizado un modelo de un solo thread, mientras que en el caso de cinco clientes PolyORB se ha configurado con un grupo de 5 threads siguiendo 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 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. En los resultados obtenidos para un cliente en Linux se puede observar que GLADE consigue mejores tiempos que PolyORB, lo que demuestra que tiene un código más ligero. El aumento de los tiempos medios en GLADE y PolyORB para cinco clientes son los esperados cuando no hay gestión de prioridades fuera de las que ofrezca el sistema operativo.

Tabla 3 Medidas en Linux para un cliente (tiempos en μs)

	Avg. Time	Max. Time	Min. Time	Std. Deviation	10% from Max. (%)
PolyORB-CORBA	1424	4302	1189	373	0.01
GLADE	415	3081	340	261	0.02
Stand-alone network	129	678	118	40	0.12

Tabla 4 Medidas en Linux para cinco clientes (tiempos en μs)

	Avg. Time	Max. Time	Min. Time	Std. Deviation	10% from Max. (%)
PolyORB-CORBA	5399	11554	1593	1050	0.02
GLADE	1700	5953	595	496	0.12

En la Tabla 5 y Tabla 6 aparecen los resultados de las medidas realizadas sobre la plataforma MaRTE OS. En la configuración utilizada en PolyORB-CORBA se ha usado el modelo sin threads para un cliente. Para el caso de cinco clientes se ha utilizado una configuración de un grupo de cinco threads con el modelo *Leader/Followers*. PolyORB-DSA, PolyORB-CORBA-RATO y RT-GLADE siguen el patrón basado en *endpoints*, es decir, la existencia de threads estáticos dedicados. El modelo de propagación de prioridades para PolyORB-CORBA ha sido el propagado por el cliente, mientras que en los otros tres casos se ha utilizado la asignación libre de prioridades y el envío únicamente del identificador de evento a través de la red. También se ha evaluado el coste temporal del uso de la red RT-EP, configurando 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).

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, obteniendo no obstante una red predecible con una dispersión menor. 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 [11] (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.

Tabla 5 Medidas en MaRTE OS para un cliente (tiempos en μs)

	Avg. Time	Max. Time	Min. Time	Std. Deviation	10% from Max. (%)
PolyORB-CORBA	2997	3012	2770	6	0.01
PolyORB-DSA	4117	4487	3835	300	42.50
RT-GLADE	1080	1151	955	23	0.03
Stand-alone network	959	964	707	3	0.01

Tabla 6 Medidas en MaRTE OS para cinco clientes (tiempos en μs)

	Avg. Time	Max. Time	Min. Time	Std. Deviation	10% from Max. (%)
PolyORB-CORBA	3527	6566	2748	727	0.11
PolyORB-DSA	4516	5299	3531	320	0.02
RT-GLADE	1000	1462	896	27	0.06

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 threads 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 6 para PolyORB con cinco clientes corresponden al menor tiempo máximo obtenido para el thread de alta prioridad, y se consiguen con todos los clientes de baja prioridad ejecutando a una prioridad superior a la intermedia. Este problema puede ser subsanado seleccionando una nueva política de control sobre el ORB, como puede ser RATO. En este caso, las medidas obtenidas tanto en el PolyORB-CORBA-RATO como en el PolyORB-DSA, utilizando una versión experimental de esta política, mejoran sensiblemente los valores de dispersión formando un conjunto más predecible.

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 los threads intermediarios.

6- CONCLUSIONES Y TRABAJO FUTURO

6.1- Trabajo realizado

En el trabajo presentado se ha realizado el 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 particular, el trabajo desarrollado para la elaboración de este proyecto se resume en los siguientes puntos:

- Análisis de los modelos de distribución actuales, identificando carencias desde el punto de vista de sistemas de tiempo real y proponiendo soluciones para ellas.
- Análisis de algunas de las implementaciones, enfatizando su gestión interna y sus posibles focos de inversión de prioridad, las entidades planificables y sus políticas de transmisión de los parámetros de planificación.
- Portabilidad de PolyORB a una plataforma de tiempo real constituida por el sistema operativo MaRTE OS y el protocolo de red RT-EP. Nuestra nueva plataforma de distribución incorpora tanto el modelo de distribución de RT-CORBA como el del DSA como se observa en la Figura 15.
- Implementación en PolyORB de una nueva política de gestión interna de peticiones remotas basada en un patrón de *endpoints*, con threads dedicados y sin envío de los parámetros de planificación a través de la red.
- Actualización de RT-GLADE a las nuevas versiones de MaRTE OS y RT-EP. De esta forma, se dispone de una plataforma uniforme (basada en las mismas versiones del sistema operativo y del protocolo de red) para todas las implementaciones.

6.2- Conclusiones

6.2.1- Middleware de distribución para tiempo real

El middleware de distribución como parte del software de un sistema de tiempo real debe 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 [6], 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 [8].

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 [8].

Las políticas de control de las peticiones remotas implementadas en PolyORB pueden servir de referencia, añadiendo un caso más en el que hay múltiples 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 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.

La libre asignación de parámetros de planificación se hace de modo genérico en RT-GLADE, mientras que 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.

6.2.2- Evaluación

En las medidas realizadas sobre la plataforma de tiempo real basada en MaRTE OS se puede comprobar como efectivamente el comportamiento del sistema es predecible, aunque tiene asociada una pequeña sobrecarga que se debe tener en cuenta. Además, el middleware basado en el DSA puro es en principio más ligero que el basado en CORBA a tenor de los datos obtenidos, siendo PolyORB-DSA más complejo debido a las transformaciones que tiene que realizar el middleware para su adaptación al núcleo de PolyORB. Asimismo, las políticas de control introducidas disminuyen la dispersión de los resultados, haciendo el conjunto más predecible.

6.3- Trabajo futuro

Nuestro trabajo va a continuar con la experimentación sobre la plataforma PolyORB, dada nuestra experiencia en el lenguaje Ada, en GLADE y en las mejoras introducidas en su comportamiento en entornos con requisitos de tiempo real. El objetivo es avanzar hacia la interoperabilidad en principio entre DSA y RT-CORBA, integrando completamente el modelo de transacciones y nuevos mecanismos de planificación para procesadores y redes. Además, las tendencias hacia la planificación flexible de los proyectos europeos FSF [2] y el actual FRESCOR [3] identifican la necesidad de dar soporte a este tipo de planificación por parte del middleware.

7- APÉNDICES

7.1- Ejemplos de uso

Para ilustrar la configuración de los equipos según la API que acabamos de describir para la política RATO se muestra a continuación un ejemplo de uso. En este caso, se realiza una operación remota para sumar dos enteros en un entorno distribuido formado por dos nodos (cliente y servidor). Este ejemplo utiliza la personalidad RT-CORBA de PolyORB aunque es fácilmente modificable para su uso con la personalidad DSA (ambas comparten la misma API de configuración).

7.1.1- Nodo cliente

Configuración del cliente

```
--  Basic setup for PolyORB
with PolyORB.Setup.Base;
--  Full Tasking profile (dynamic priorities must be included)
with PolyORB.Setup.Tasking.Full_Tasking;
--  ORB Policy
with PolyORB.ORB.Thread_Per_Target;
--  ORB Control Policy
with PolyORB.ORB_Controller.Ready_To_Go;

-- Personalities setup
-- GIOP --
with PolyORB.Binding_Data.GIOP;
--  RTEP Personality
with PolyORB.Setup.RTEP_MAC;
with PolyORB.Binding_Data.GIOP.RTEP_MAC;
--  RATO-RTEP
with PolyORB.Transport.RTEP_Common.RATO_RTEP_MAC;

--  System Configuration
with Configuration_File;

package body Client_Configuration is
begin
   --  Internal threads in PolyORB
   Configuration_File.Num_Threads_In_Pool := X;
   --  Naming service
   Configuration_File.Naming_Service_IOR :=
      new String'("IOR:XXX");
end Client_Configuration;
```

Programa principal del cliente

```
--  Corba, MaRTE OS, Exceptions, remote operation stubs
...

--  System Configuration
with Client_Configuration;
pragma Elaborate_All (Client_Configuration);
```

```
--  API
with PolyORB.API.RTEP_MAC.Scheduling.Priorities;
with PolyORB.API.RTEP_MAC;    use PolyORB.API.RTEP_MAC;

procedure Client is

   --  Main task priority (check value in MaRTE OS)
   pragma Priority (Z);

   --  Variables
   ...
   --  Command line args
   ...
   --  Local variables
   ...

begin

   --  Debug facilities
   PolyORB.API.RTEP_MAC.Enable_Debug (False);

   idorb := CORBA.ORB.To_CORBA_String ("ORB");

   --  Init ORB
   CORBA.ORB.Init (idorb, Argv);

   --  Recover IOR from file (FAT not available in MaRTE)
   CORBA.ORB.String_To_Object
        (CORBA.To_CORBA_String (IOR_Suma),
         Ior_Ref);

   --  ******************** RTEP ***************************
      declare
         P            : Sched.Priorities_Message_Scheduling_Parameters;
         P_Ref        : Sched.Priorities_Message_Scheduling_Parameters_Ref;
         Endpoint_Out : RTEP_MAC.Endpoint_Id;
      begin

         --  Scheduling API
         P.P_in := RTEP.Priority (X);
         Sched.Set_Scheduling_Params (P, P_Ref);

         --  Set up endpoint to send requests
         Sched.Create_Send_Endpoint
           (Params    => P_Ref,
            Event_ID  => Id,
            Dest_Node => Station,
            Channel   => Channel,
            Endpoint  => Endpoint_Out);

      end;
      --  ******************** RTEP ***************************

--  Remote operation
Res := Suma.SumaNum (Ior_Ref, Num1, Num2);


exception
   when E : others =>
      Kernel_Console.Put ("Received exception: ");
      Kernel_Console.Put (Ada.Exceptions.Exception_Information (E));

end Client;
```

Configuración del servidor

```
--  ORB Policy
with PolyORB.ORB.Thread_Per_Target;
--  ORB Control Policy
with PolyORB.ORB_Controller.Ready_To_Go;
--  Full Tasking profile (dynamic priorities must be included)
with PolyORB.Setup.Tasking.Full_Tasking;
--  RT-POA
with PolyORB.Setup.OA.Basic_RT_POA;

--  Personalities setup
--  GIOP + RTEP
with PolyORB.Setup.RTEP_MAC;
--  RATO-RTEP
with PolyORB.Setup.Access_Points.RATO_RTEP_MAC;

--  System Configuration
with Configuration_File;

package body Server_Configuration is

begin
   --  Internal threads in PolyORB
   Configuration_File.Num_Threads_In_Pool := X;
   --  Naming service
   Configuration_File.Naming_Service_IOR :=
      new String'("IOR:XXX");
end Server_Configuration;
```

Programa principal del servidor

```
--  MaRTE OS, Exceptions, RT-CORBA
...

--  System Configuration
with Server_Configuration;
pragma Elaborate_All (Server_Configuration);

--  API
with PolyORB.API.RTEP_MAC.Scheduling.Priorities;

procedure Server is

   --  Main task priority: It must be less than created task to serve requests
   --  to allow prior creation
   pragma Priority (Z);

   --  Variables
   ...
   --  Command line arguments
   ...
```

begin

```
   --  Debug facilities
   PolyORB.API.RTEP_MAC.Enable_Debug (False);

   --  ORB Init
   idorb := CORBA.ORB.To_CORBA_String ("ORB");
   CORBA.ORB.Init (idorb, Argv);
```

```
   --  Create remote object
   Object := new Suma.Impl.Object;

   --  Subprogram to initialize RTPOA (following CORBA standard)
   Inicio.Servidor_RTCorba
     (Object => Object,
      RT_POA => RT_POA,
      IOR    => IOR_ref);


   --  ******************** RTEP ***************************

   declare
      P            : Sched.Priorities_Message_Scheduling_Parameters;
      P_Ref        : Sched.Priorities_Message_Scheduling_Parameters_Ref;
      Endpoint_Out : RTEP_MAC.Endpoint_Id;
      Node_To_Reply : Station_ID := YYY;

   begin

      --  Scheduling API
      P.P_in := RTEP.Priority (X);
      Sched.Set_Scheduling_Params (P, P_Ref);

      --  Setting receive endpoints
      Sched.Create_Receive_Endpoint
           (Params    => P_Ref,
            Dest_Node => Node_To_Reply,
            Event_ID  => Id,
            Channel   => Channel,
            Endpoint  => Endpoint_Out);

   end;

   --  ******************** RTEP ***************************

   --  We need to rewrite the IOR because AP is not created at init phase
   IOR_ref :=
      PortableServer.POA.Servant_To_Reference
        (PortableServer.POA.Local_Ref (RT_POA),
         PortableServer.Servant (Object));


   --  Launch the server
   CORBA.ORB.Run;

exception
   when E : others =>
      Put ("Received exception: ");
      Put (Ada.Exceptions.Exception_Information (E));

end Server;
```

7.2- Paquetes complementarios

7.2.1- Debug facilities

Debido a que MaRTE OS todavía no dispone de un sistema de ficheros totalmente operativo, se ha desarrollado un paquete básico de depuración para hacer este paso independiente de la compilación.

El paquete *PolyORB.API.RTEP_MAC* incluye la API para gestionarlo. Simplemente se trata de un procedimiento denominado *Enable_Debug* que activa/desactiva distintos segmentos del software desarrollado.

```
procedure Enable_Debug (VARIABLES);

Enable_Debug : Activa/desactiva las funciones de depuración por la salida es-
tándar a través de la aplicación

Variables a configurar:
Transport : Información a nivel de red: Destino, Canales, Prioridades...
Asynch : Registro de las fuentes de eventos en el ORB
Lanes  : Comportamiento interno del pool de threads en la RT-POA
Leader_Followers : Comportamiento interno del thread en la política L&F
RATO : Comportamiento interno del thread en la política RATO
QoS  : Información sobre los parámetros de QoS relacionados con la RT-EP (em-
paquetado/desempaquetado de los Service Context)
TPT  : Comportamiento interno del thread en la política TPT
Init : Detalles del proceso de inicialización
References : Información sobre la reutilización del binding a objetos
Buffers : Detalles sobre el almacenamiento de datos
```

Ejemplo de uso: Para activar las características de depuración de transporte, realice la siguiente llamada

```
Enable_Debug (Transport => True);
```

7.2.2- Servicio de Nombres

El servicio de nombres permite la asociación de referencias a objetos con nombres más sencillos e intuitivos. De esta forma, los clientes pueden conseguir la referencia a un objeto CORBA utilizando un nombre, totalmente independiente de la distribución física del sistema. Además, los nombres presentan una organización jerárquica. El esquema de funcionamiento se muestra en la Figura 16.

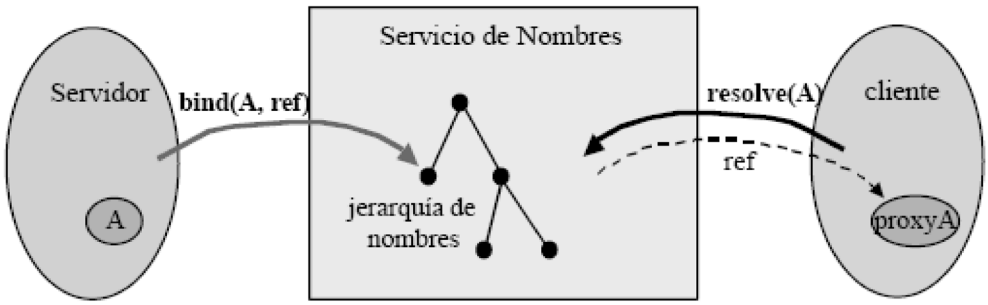

Figura 16 Funcionamiento del servicio de nombres en CORBA

Este es un servicio de CORBA que ha sido portado a MaRTE OS ya que resulta estrictamente necesario para la personalidad DSA de PolyORB. Asimismo, se ha configurado para el uso de la red RT-EP. De cara al usuario final, las interfaces que debe manipular son:

PolyORB.Corba_p.MaRTE_OS: Ya que no se dispone de un sistema de ficheros para pasar el IOR del propio servidor de nombres, hemos añadido una forma de enviar el IOR desde el ejecutable a través del fichero de configuración (ver sección posterior), haciendo este paso independiente de la compilación.

PolyORB-Api-IOR_Tools: Interfaz de acceso a la información contenida en el perfil de protocolo de la RT-EP asociada al IOR obtenido a través del servidor de nombres.

7.2.3- Fichero de Configuración

Este fichero (*configuration_file.ads*) nos permite configurar algunos parámetros en tiempo de inicialización de la aplicación, al estilo del *polyorb.conf* existente en el software original. Sin embargo, dada la imposibilidad de utilizar un sistema de ficheros que almacene dicha configuración, ésta debe ser integrada en la aplicación cargada en memoria. Los parámetros configurables son:

- *Num_Threads_In_Pool*: Número de threads que puede haber en el pool. En el caso del pool anónimo, especifica el número de threads creados al inicio. En el caso de la política *Thread Per Target* no se usa.
- *Num_Monitors*: Máximo número de monitores en el sistema. Su valor debería corresponder con Num_Threads_In_Pool + 1 en el caso de TPT (un thread por monitor y un thread de background). Este valor es crítico en cuanto a rendimiento por lo que debe configurarse de forma precisa.
- *Naming_Service_IOR*: Define el Interoperable Object Reference del servidor de nombres. A través de este parámetro, nuestro servidor podrá registrar sus objetos remotos y el cliente obtendrá el *IOR* de dicho objeto.

7.2.4- Utilidades de medidas de tiempo

Para la realización de las medidas del software se ha desarrollado un paquete con diversas utilidades para facilitar su uso. Su funcionamiento se basa en un identificador de medidas que permite segmentar el código de manera intuitiva. Se divide en dos partes bien diferenciadas según dónde se redireccionen los resultados.

- Salida por pantalla

  Es la forma más sencilla de obtener resultados. Esta API incluye los siguientes procedimientos:

```
procedure Set_And_Reset_Time_Measures (Max_Number : in Natural);
```

Set_And_Reset_Time_Measures : Especifica el número total de medidas que se deben realizar y resetea todas las variables temporales
Max_Number : Número total de medidas

```
procedure Take_Measure_Start (Id : in Natural);

Take_Measure_Start : Especifica el punto de comienzo de la medida ID
Id : Identificador de la medida

procedure Take_Measure_Stop (Id : in Natural);

Take_Measure_Stop : Especifica el punto final de la medida ID
Id : Identificador de la medida

procedure Store_Measure (Id : in Natural;
                         Max_Time_Allowed: in Time_Span;
                         Top_Measures_Reached : out Boolean);

Store_Measure : Almacena la medida Id, comprobando si ésta es válida y si el
límite de medidas a realizar se ha alcanzado
Max_Time_Allowed : Permite establecer un límite para considerar la validez de
las medidas. Evita problemas a nivel de hardware, como errores de red
Top_Measures_Reached : Parámetro que indica que se ha alcanzado el número de
medidas deseadas

procedure Show_Average (Id : in Natural);

Show_Average : Imprime las figuras de tiempo de la medida ID almacenada pre-
viamente: media, máximo y mínimo obtenidos
Id : Identificador de la medida

procedure Show_Std_Deviation (Id : in Natural);

Show_Std_Deviation : Imprime la desviación estándar y el número de medidas
incluídas en el intervalo entre el máximo y el 10% de la diferencia entre el
máximo y el valor medio
Id : Identificador de la medida
```

- Salida por la red

  Una forma más conveniente de realizar las medidas si el objetivo es almacenarlas de forma persistente se presenta en esta segunda parte. Esta interfaz es simplemente una envoltura para Ada de las funciones temporales de POSIX incluidas en MaRTE OS que utilizan la red ethernet como medio para transmitir los resultados. A diferencia del modo consola, estas medidas son identificadas a través de un nombre.

```
function Reset_And_Init_Eth_Time_Measures (Name : in String) return Integer;

Reset_And_Init_Eth_Time_Measures : Resetea todas las variables temporales y
devuelve un identificador asociado al nombre de la medida
Name : Nombre que identificará la medida en el nodo destino de los datos

procedure Take_Eth_Measure_Start (Id : in Integer);

Take_Eth_Measure_Start : Especifica el punto de comienzo de la medida ID
Id : Identificador de la medida

procedure Take_Eth_Measure_Stop (Id : in Integer);

Take_Eth_Measure_Stop : Especifica el punto final de la medida ID
Id : Identificador de la medida

procedure Send_Data;

Send_Data : Envía los datos almacenados al nodo destino
```

Finalmente, en el nodo receptor de datos deberá ejecutarse el software de recepción para poder almacenar los datos en tu computadora

- Ej: La aplicación *linux_eth_receive_log* incluida en la distribución de MaRTE OS

8- BIBLIOGRAFÍA

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

[2] M. Aldea, G. Bernat, I. Broster, A. Burns, R. Dobrin, J.M. Drake, G. Fohler, P. Gai, M. González Harbour, G. Guidi, J.J. Gutiérrez, T. Lennvall, G. Lipari, J.M. Martínez, J.L. Medina, J.C. Palencia, and M. Trimarchi. "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] FRESCOR project web page: http://frescor.org

[4] J.J. Gutiérrez, and M. González Harbour. "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, June 1999.

[5] Y. Krishnamurthy, I. Pyarali, C. Gill, L. Mgeta, Y. Zhang, S. Torri, and D.C. Schmidt. "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 (Canada), May 2004.

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

[7] J. López Campos, J.J. Gutiérrez, and M. 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, June 2004.

[8] J. López Campos, J.J. Gutiérrez, and M. 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, June 2006.

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

[10] J.M. Martínez, and M. González Harbour. "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, June 2005.

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

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

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

[14] 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.

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

[16] I. Pyarali, M. Spivak, D.C. Schmidt, and R. Cytron. "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, June 2001.

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

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

[19] S. Tucker Taft, Robert A. Duff, Randall L. Brukardt, Erhard Ploedereder, and Pascal Leroy (Eds.). "Ada 2005 Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1". LNCS 4348, Springer, 2006.

[20] T. Vergnaud, J. Hugues, L. Pautet, and F. Kordon. "PolyORB: a Schizophrenic Middleware to Build Versatile Reliable Distributed Applications". Proc.of the 9th International Conference on Reliable Software Technologies, Palma de Mallorca (Spain), in LNCS, Vol. 3063, June 2004.

[21] Jonathan S. Anderson and E. Douglas Jensen. The distributed real-time specification for Java: A status report. In Proceedings of the 4th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES 2006), page UNKNOWN. October 2006. Conservatoire National des Arts et Métiers (CNAM) Paris, France, 11-13 October 2006.

[22] Bechir Zalila, Jérôme Hugues, Laurent Pautet. An improved IDL compiler for optimizing CORBA applications. Proceedings of the 2006 annual ACM SIGAda international conference on Ada, Albuquerque, New Mexico, USA.

[23] Yamuna Krishnamurthy, Irfan Pyarali. Design and Implementation Issues in the Dynamic Scheduling Real-Time CORBA 2.0 Specification OOMWorks, LLC

[24] POSIX.13 (1998). *IEEE Std. 1003.13-1998. Information Technology - Standardized Application Environment Profile - POSIX Realtime Application Support (AEP)*. The Institute of Electrical and Electronics Engineers.

[25] 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,

[26] William Stallings. *"Comunicaciones y redes de computadores"*, Prentice Hall, 2004.

[27] M.H. Klein, T. Ralya, B. Pollak, R. Obenza, M.G. Harbour, "*A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*", Kluwer Academic Publishers, 1993

[28] Juan López Campos, J. Javier Gutiérrez and M. González Harbour. RT-GLADE: Implementación Optimizada para Tiempo Real del Anexo de Sistemas Distribuidos de Ada 95. XII Jornadas de Concurrencia y Sistemas Distribuidos, Las Navas del Marqués (Ávila), Spain, June 2004.