Simposio de Informática y Telecomunicaciones SIT'02

Implementación de un ORB para Dispositivos Empotrados: Un Caso de Estudio*

Julián de la Morena Borja, Francisco Moya Fernández,
Juan Carlos López López

Escuela Superior de Informática de Ciudad Real
Universidad de Castilla-La Mancha,
julian.delamorena@uclm.es
Francisco.Moya@uclm.es
JuanCarlos.Lopez@uclm.es

Resumen El desarrollo de un ORB para dispositivos de capacidad de cómputo reducida plantea serios problemas si queremos ajustarnos al estándar de CORBA, implementando todas las características definidas en él. En este artículo se describe un ORB implementado para uno de estos dispositivos, como es una placa TINI. El trabajo principal se centra en el análisis de la funcionalidad que debamos dar al ORB, dejando aquellas caracteriscas básicas para la ejecución de la mayoría de aplicaciones CORBA. Se busca aprovechar al máximo las características de este dispositivo, ajustándonos a sus propias limitaciones.

1 Introducción

En proyectos donde CORBA vaya a ser la plataforma de comunicaciones entre los distintos elementos que componen la infraestructura, puede ser necesaria, por razones tanto económicas como de tamaño, la utilización de pequeños dispositivos de cómputo frente a otros elementos de mayor capacidad, como ordenadores personales. Las capacidades de cómputo de dichos dispositivos distan mucho de las necesidades mínimas necesarias para la utilización de un ORB que reúna todas, o la mayoría, de características definidas en el estándar del OMG [2].

Ante esta circunstancia, una serie de empresas, como Alcatel, Hewlett Packard, Inprise Corporation, Iona Technologies y Northern Telecom definieron una versión reducida de CORBA, *minimumCORBA*[3]. En esta versión sólo han quedado definidas aquellas características que se consideran imprescindibles para la ejecución de la mayoría de aplicaciones que se realizan en CORBA.

Aunque *minimumCORBA* reduce considerablemente los requisitos respecto a la especificación completa de CORBA, en entornos donde los recursos son limitados, algunas de las características que se han mantenido no son necesarias para la aplicación que vamos a dar al ORB, sobrecargando innecesariamente el entorno. Por lo tanto, antes de implementar un ORB en estas condiciones de

* Este trabajo ha sido financiado por el Ministerio de Ciencia Tecnología (TIC2000-0583-C02-01) y la Junta de Comunidades de Castila-La Mancha (PBI-02-024)

cómputo se tuvo que analizar cuidadosamente que características deseamos que posea nuestro ORB y cuales no resultan estrictamente necesarias.

1.1 Entorno Domótico

La necesidad de utilizar dispositivos de baja capacidad de cómputo surgió a la hora de querer utilizar CORBA como plataforma de comunicaciones dentro de un entorno domótico.

Esto se enmarca dentro del proyecto SENDA [4], emprendido por el grupo de Arquitectura y Redes de Computadores (ARCO) de la Universidad de Castilla-La Mancha. Este proyecto busca el desarrollo de servicios domóticos, utilizando CORBA como la plataforma de comunicaciones para los dispositivos que podemos encontrar en un hogar común.

En estos entornos, la utilización masiva de PCs controlando los dispositivos de toda la casa, se convierte en una tarea inviable, debido al tamaño que un PC ocupa, junto con sus periféricos de E/S, como monitor, teclado, etc. Además hay que añadir el costo que esto supondría para una correcta instalación, añadiendo los costes de mantenimiento y administración que necesita un PC. Estas razones obligaron a la búsqueda de dispositivos que permitiesen realizar las mismas tareas que haría un PC, pero con un coste económico, de tamaño y mantenimiento considerablemente inferior.

1.2 Placa TINI

Dentro de estos dispositivos empotrados que son utilizados en la infraestructura SENDA, se encuentran placas TINI (*Tiny Internet Interface*), desarrolladas por Dallas Semiconductor [7]. Estas placas presentan enormes posibilidades para los desarrolladores por diferentes razones. La primera, y principal, es que ejecuta programas Java, al disponer de una máquina virtual de Java, lo cual permite la utilización de un lenguaje de alto nivel para el desarrollo de nuestras aplicaciones. Además tenemos el factor a tener en cuenta de que el coste de una de estas placas no supera los 70$. Otra de las grandes ventajas que presentan, y que las hacen muy interesantes en el entorno donde se quieren utilizar, es que diponemos de facilidades a la hora de interconectarlas, ya que disponen de interfaces RS-232, CAN y Ethernet.

Sin embargo, el TINI no está exento de problemas y éstos vienen principalmente del hecho de que sea un dispositivo de pequeño tamaño. Así tenemos una memoría limitada, ya que tan sólo disponemos de 1MiB[1] de memoria, de los cuales 120KiB[2] ya se encuentran utilizados por el Shell del sistema. Además, la máquina virtual que corre el TINI soporta un API limitado, no estando presentes incluso aquellas clases que se consideran Java puro, lo que limita ciertos aspectos de funcionalidad que resultarían muy interesantes a nuestro ORB.

[1] mebibyte
[2] kilibyte

1.3 Alternativas de Objetos Distribuidos

Existen otro tipos de tecnologías que nos permiten el uso de objetos distribuidos, como puede ser RMI. RMI se trata de un mecanismo de invocación dinámica de métodos pensado para su uso en entornos de ejecución Java, como es una placa TINI.

En un principio, la complejidad de implementación de RMI debería ser menor si se dispusiese de un API adecuado, sin embargo, el API que lleva la Java Virtual Machine del TINI sólo lleva implementada una pequeña versión del API jdk 1.1. Este API es escaso para la implementación de las clases de RMI, ya que mecanismos como la serialización de objetos, parte fundamental de RMI, no se encuentra presente en el API del TINI.

Además la utilización de RMI supondría que la interacción de nuestra placa tendría que realizarse con dispositivos que dispusiesen de una máquina virtual de Java, hecho que no se da en el entorno donde se pretende utilizar.

Estas restricciones, que en un principio nos impondría utilizar RMI, y las cuales no se presentan utilizando CORBA, y la búsqueda de la mayor capacidad de adaptación de nuestro dispositivo al entorno donde vaya a ser utilizado, condujo a la implementación del TINIORB por delante de la posibilidad de la implementación de una versión de RMI para este dispositivo.

1.4 *minimumCORBA*

Esta versión de CORBA fue especialmente pensada para ser utilizada en dispositivos con unos recursos de cómputo limitados. En ella quedaron definidas aquellas características que se consideran básicas a la hora de desarrollar aplicaciones CORBA.

La principal razón para definir esta versión de CORBA es que algunas aplicaciones CORBA son demasiado grandes en términos de tamaño y necesitan demasiados recursos, dos factores que están muy limitados a la hora de desarrollar aplicaciones en entornos empotrados. Esto provocó la necesidad de realizar una versión de CORBA que eliminara algunas de las características que consumían estos recursos. Todo lo relacionado con los CORBAservices, CORBAsecurity, etc, se consideran extensiones opcionales a minimumCORBA, y, por lo tanto, no resulta imprescindible su implementación.

Algunas de las características de CORBA que han sido omitidas tienen importancia en algunas aplicaciones CORBA. Sin embargo, añaden un coste, en términos de recursos, que no justifican su inclusión ya que no son utilizados estos servicios por la mayoría de las aplicaciones. En el caso de querer ser utilizadas estas características por alguna aplicación, deberán ser implementadas para ese caso concreto.

A pesar que *minimumCORBA* busca adaptarse a los entornos empotrados, aún incluye características del ORB que no son necesarias para entornos como en el que consideramos en este trabajo. Por esta razón hubo que analizar que características, de las que presenta, son interesantes para la funcionalidad que queremos para TINIORB.

1.5 ORB para Dispositivos Empotrados

Existen ORBs que han sido también desarrolados para su aplicación en entornos de capacidad de cómputo limitada. e*ORB [6] desarrollado por Vertel es un ORB que ha sido pensado para su utilización en entornos empotrados y de tiempo real. Han utilizado para su desarrollo la especificación minimumCORBA, existiendo versiones en distintos lenguajes, entre ellos Java, en el que sólo han utilizado las primitivas del Java puro para buscar la mayor portabilidad posible.

Otro ejemplo de ORB para este tipo de entornos es ROFES [8], buscando una implementación de Real-Time CORBA.

Estos ejemplos de ORB han sido pensados para su utilización en entornos generales, con una serie de requisitos que se adaptan la mayoría de dispositivos empotrados. e*ORB podría ser una alternativa para su utilización en un entorno como es el TINI, sin embargo utiliza primitivas de Java, que aún siendo generales, no se encuentrar en el API de nuestra máquina virtual. Este proyecto no buscó un ORB para el uso en el mayor número de dispositivos posibles, sino que se adaptase para su utilización eficiente en la placa TINI, aprovenchando todas las posibilidades que nos ofrece, sobre todo a la hora de su interconexión con distintas redes de comunicaciones.

2 Objetivos de TINIORB

El objetivo era desarrollar un ORB que permitiera integrar nuestras placas TINI dentro de una infraestructura domótica concreta, donde CORBA ha sido elegida como plataforma de comunicaciones. Para dicho desarrollo se ha tomado como punto de partida JacORB [5], ORB desarrollado integramente en Java.

Sin embargo el uso de memoria que genera JacORB, 8 MiB para empezar a ejecutar el ORB y una media de 19MiB para una aplicación CORBA normal, y la utilización de clases que no se encuentran en el API del TINI, hace practicamente irrealizable una adaptación directa del JacORB al TINI.

La implementación de TINIORB tiene como objetivo la utilización de nuestra placa como pasarela residencial entre los dispositivos que controlamos directamente a través de los puertos de E/S que dispone nuestro dispositivo, y el resto de la red que compone la infraestructura.

3 Funcionalidad del ORB

El primer paso que se realizó es un análisis de las características mínimas del ORB para su correcta integración dentro de la infraestrauctura donde se quiere utilizar.

El método de invocación y recepción de peticiones será mediante invocación estática, utilizando los interfaces IDL. A partir de estos interfaces IDL, y mediante un compilador adaptado a este ORB, se generarán los stubs y esqueletos necesarios para la aplicación CORBA concreta.

3.1 Protocolo de Comunicaciones

Un punto básico que debe cumplir todo ORB es la implementación de un protocolo de comunicaciones. El OMG, a la hora de definir CORBA, y para conseguir la interoperabilidad entre los ORBs de distintos proveedores, definió un protocolo de comunicaciones independiente tanto del ORB, como de la máquina o del protocolo de transporte utilizado. Este protocolo se llamó GIOP (General Inter-ORB Protocol). En este proyecto, se ha utilizado más concretamente IIOP, que es la versión de GIOP para TCP/IP.

Uno de los principales objetivos que se buscó al definir GIOP, fue la simplicidad, para asegurar la compatibilidad. Se buscó que fuese lo más independiente posible del resto del ORB, para que los cambios futuros sobre el protocolo no influyesen de manera decisiva sobre el resto de la funcionalidad. De esta manera, se define una estructura de capas, donde el ORB estaría por encima de GIOP y por encima del ORB se encontraría la aplicación CORBA concreta.

Si se quiere que el ORB sea interoperable con el resto de ORBs se debe respetar escrupulosamente la especificación de GIOP [2]. En ella se especifica claramente cómo debe ser el formato de los mensajes que se intercambian a la hora de una petición, así como el de los datos intercambiados. Este formato de los datos es independiente tanto de la arquitectura del cliente como del servidor, siendo el ORB el encargado de la adaptación de esta representación independiente a la representación concreta de la máquina donde se esté ejecutando.

En este ORB se ha implementado la versión de IIOP 1.0.

3.2 Manejo de Objetos

Una de las principales características de los objetos COBRA, es la de poder pasar las referencias de éstos entre los distintos ORBs, siendo capaces todos ellos de poder interpretarlas y acceder a los objetos. Estas referencias pueden ser representadas mediante los IOR, cadenas de texto que contienen toda la información necesaria para poder acceder al objeto que referencia.

Cada objeto que se encuentre en el TINI tendrá un único *profile* de acceso mediante IIOP, conteniendo el IOR generado para cada objeto la información para poder acceder al mismo. En este caso contiene tanto la dirección de la máquina donde se encuentra el objeto, como el puerto de acceso para realizar las peticiones.

Además disponemos de las operaciones básicas en objetos, como la creación y eliminación de objetos, referencias nulas, equivalencia entre objetos, etc.

3.3 POA

Para la recepción de peticiones por parte de los objetos CORBA que se encuentren corriendo en el ORB, se necesita la implementación del adaptador de objetos, más concretamente del POA. Sin embargo, no se puede implementar todas la características definidas en el estándar de CORBA debido a las limitaciones del entorno de ejecución, por lo que habrá que ceñirse a las necesidades particulares de la infraestrutura concreta.

Una de las primeras labores a realizada fue la definición de las políticas que el POA admitiría. En este caso sólo se implementaron aquellas políticas más comúnmente utilizadas en la mayoría de aplicaciones CORBA, teniendo además en cuenta no sobrecargar en exceso los recursos disponibles. Se describen a continuación.

Thread Policy Se selecciona el modelos de hilos para servir las peticiones recibidas a los objetos que tenemos registrados. La política elegida ha sido el *ORB_CTRL_MODEL*, en la cual el ORB es el encargado de adaptar el modelo de hilos dependiendo de la implementación del ORB. La posibilidad de poder atender peticiones de manera concurrente fue limitada al no disponer de más de 16 hilos de ejecución por proceso, por lo que normalmente se atenderá una sóla petición en cada instante, encolando el resto que vayan llegando.

Lifespan Policy En ella se define la vida que tendrían los objetos que se registren bajo el POA. Se ha optado por la implementación de la política *TRANSIENT*, donde los objetos sólo pueden invocarse mientras el proceso que los ha creado continúe activo.

IdAssignment Policy Cada objeto debe tener asignado un identificador único dentro del POA. La elección puede ser realizada por el usuario, o bien, de forma automática por el POA. Ambas posibilidades están contempladas.

IdUniqueness Policy Se tiene la posibilidad de asignar varios identificadores de objetos a un solo *servant*, facilitando la gestión de recursos en caso de tener un gran número de objetos.

RequestProcessing Policy Para la asociación de peticiones a sirvientes se ha elegido la política de utilizar el *Active Object Map* (AOM), en la cual se mantiene una tabla que asocia objetos con sirvientes.

ServantRetention Policy En este caso los *servants* de los objetos permanecerán activos en memoria, lo que limitará el número de objetos que podremos utilizar concurrentemente.

3.4 Servicio de Nombres

Se ha incluido la posibilidad de que las aplicaciones que vayan a ser ejecutadas bajo este ORB puedan realizar las operaciones propias de un cliente del interfaz `CosNaming::NamingContext` definido por OMG. De esta forma cualquier objeto podrá registrarse en un servicio de nombres, además de resolver cualquier referencia del objeto que se encontrase registrado en él.

4 Ejemplo de Uso

En el caso del proyecto SENDA, se utiliza una placa TINI para el control de una serie de dispositivos X10 a través de un interfaz CM11 conectado al TINI a través de su puerto serie. X10 nos permite el control de dispositivos domóticos mediante la transmisión de comandos utilizando la red eléctrica común, y mediante el CM11 podremos mandar comandos a dicha red a través del puerto RS-232.

SENDA proporciona un interfaz CORBA para cada dispositivo, abstrayendo completamente los detalles tecnológicos y de control. Un pequeño número de interfaces genéricos permite el control, por parte de aplicaciones independientes, de todos los dispositivos que podamos encontrarnos normalmente en un hogar.

De esta manera en el TINI existirán tantos objetos CORBA como dispositivos X10 pueda controlar a través de interfaz CM11 que tenemos conectado. Así conseguimos que el resto de los elementos que componen la red domótica puedan controlar estos dispositivos X10.

5 Rendimiento

Este ORB ha sido probado en el entorno que ha sido descrito en el apartado anterior. En dicho entorno ha debido interactuar con ORBs como MICO, ORBIT, JacORB1.3. Con ninguno de estos ORBs ha tenido ningún problema de comunicaciones, permitiéndose recepción de invocaciones por parte de clientes externos y realizando peticiones por parte de los objetos que se encuentran en nuestra placa hacia objetos implementados en otros ORBs.

El tamaño del ejecutable, con toda la implementación correspondiente al ORB para la recepción y envío de invocaciones remotas, más la implementación necesaria para el control de los dispositivos domóticos, como se ha explicado anteriormente, con 5 objetos CORBA ha ocupado 111 KiB. En ejecución se consumen 230KiB, dejando libres alrededor de 630KiB de la memoria disponible. Tan sólo se necesita en ejecución un socket abierto para la recepción de peticiones.

En estas condiciones, los recursos de nuestro dispositivo no se encuentran al límite. Sin embargo, lo mayores problemas que hemos encontrado durante la ejecución son de velocidad del procesador, ya que las peticiones recibidas tienen un retardo de aproximadamente 2 segundos.

6 Desarrollo Fufuro

Se están implementando las características de invocación dinámica que ofrece CORBA. Uno de los principales puntos que deben desarrollarse es la utilización de los tipos Any, que nos permitirán una mayor flexibilidad de invocaciones entre clientes y servidores.

También se está implementado la posibilidad de utilización de objetos persistentes, ya que pueden ser de especial utilidad en los entornos domóticos en los que estamos trabajando.

Por último, se pretende desarrollar versiones reducidas de algunos de los servicios estándares de CORBA, como el servicio de propagación de eventos.

Teniendo en cuenta que este ORB utiliza primitivas Java que se encuentran en todos los entornos Java, podría tenerse en cuenta para utilizarse en otro tipo de entornos, como podría ser su inclusión en Applets, debido a la facilidad que tendría en transmitirse por la web por su escaso tamaño.

Referencias

1. Henning, M., Vinoski, S. Programación Avanzada en CORBA (2002)
2. OMG The Common Object Request Broker: Architecture and Specefication (1998) ftp://ftp.omg.org/pub/formal/98-12-01.pdf
3. OMG minimumCORBA (1998) ftp://ftp.omg.org/pub/orbos/98-08-04.pdf
4. Francisco Moya Fernández, Félix Jesús Villanueva Molina, Juan Carlos López López SENDA: Hacia una infraestructura domótica global XI Jornadas TELECOM I+D (octubre 2001)
5. http://www.jacorb.org
6. Vertel e*ORB Technical Description http://www.vertel.com/products/eorb.asp
7. http://www.ibutton.com/TINI/index.html
8. http://www.lfbs.rwth-aachen.de/users/stefan/rofes/