Un modelo de soporte a interoperabilidad de aplicaciones distribuidas sobre ambientes CORBA y DCOM

*Camilo Villarreal**
*Jorge Enrique Jiménez***

Resumen: *Las tecnologías de objetos distribuidos que han logrado consolidarse con mayor éxito en el mercado, CORBA y DCOM, han permitido integrar aplicaciones distribuidas construidas sobre plataformas y lenguajes tradicionalmente incompatibles, logrando un mejor aprovechamiento de recursos y aumentando la eficiencia de los sistemas de información. Sin embargo, la incompatibilidad entre estas dos tecnologías impide que la interoperabilidad de aplicaciones sea completa, puesto que cada vez se encuentran con mayor frecuencia sistemas de información que conviven en ambos mundos. Se propone un modelo para interconectar aplicaciones pertenecientes a ambientes distintos en forma bidireccional.*

Abstract: *The Distributed Object Technologies CORBA and DCOM have gained great success in the market and have been positioned as the most accepted models for integration of distributed applications constructed on traditionally incompatible platforms and programming languages, allowing for a better use of resources and leveraging efficiency of information systems. However, incompatibility between these two environments, restricts interoperability between distributed applications considering that information systems coexisting in both worlds are more commonly found these days. In this paper, a model is proposed to allow interconnection of applications that belong to the two different environments.*

* Ingeniero de Sistemas y Magíster en Ingeniería de Sistemas, Universidad de los Andes. Profesor Instructor, Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana.

** Estudiante de último semestre de Ingeniería de Sistemas.

1. Introducción

El problema de integrar aplicaciones que están inmersas en los modelos distribuidos mutuamente incompatibles CORBA y DCOM ha surgido a partir de la necesidad de acceder a servicios ofrecidos por componentes que pertenecen a uno de estos ambientes, desde un cliente que se encuentra en el ambiente contrario.

Al no poder invocar operaciones de componentes remotos que pertenecen a otro ambiente, el conjunto de servicios que un cliente puede usar está restringido a las funcionalidades que le ofrecen los objetos servidores de su propio ambiente distribuido y esto limita la integración de los sistemas de información. En el estado del arte se encuentran soluciones propietarias a este problema que exigen la instalación de productos adicionales que cumplan con las especificaciones de un fabricante particular, elevando los costos en forma exagerada. Adicionalmente, por lo general, una solución propietaria bloquea el uso de productos diferentes a los provistos por el vendedor, atando los nuevos desarrollos a sus especificaciones y generando dependencia en el usuario. [Dolgicer y Fischer, 1997] En este documento se propone un modelo de interconexión bidireccional de objetos pertenecientes a ambientes diferentes que permite la invocación de servicios remotos que usan tipos de datos básicos en sus parámetros y valores de retorno, cuya implementación es relativamente sencilla de llevar a cabo y no está atada a una familia de productos en particular.

La única herramienta que el modelo exige usar es la implementación CORBA Java IDL (de *Sun Microsystems*), seleccionada por razones de flexibilidad de desarrollo, portabilidad entre plataformas y por ser de libre distribución.

2. Estructura del modelo

El diseño del modelo se ha definido teniendo en cuenta que cuando se construyen aplicaciones distribuidas sobre ambientes *middleware*, los objetos pueden estar implementados sobre diferentes lenguajes de programación. Además, para el programador de la parte cliente, una invocación a un servicio residente en el ambiente opuesto debe ser natural en lo posible, es decir, debe tener la misma apariencia que un llamado a un servicio en su propio ambiente. Para lograr trasparencia y uniformidad, el modelo ha sido estructurado en dos grandes partes donde cada una implementa un mecanismo de conexión unidireccional.

2.1 Interconexión DCOM-CORBA

Esta parte del modelo establece los mecanismos necesarios para lograr que un cliente DCOM pueda invocar servicios de un objeto servidor CORBA. Está compuesto por dos módulos, *servidor receptor* y *servidor de invocación*, que se encar-

gan de capturar los requerimientos en el ambiente local y realizar la invocación requerida en el ambiente opuesto, respectivamente. (Figura 1) A nivel global, en el escenario de interconexión unidireccional DCOM-CORBA, se identifican los siguientes participantes:

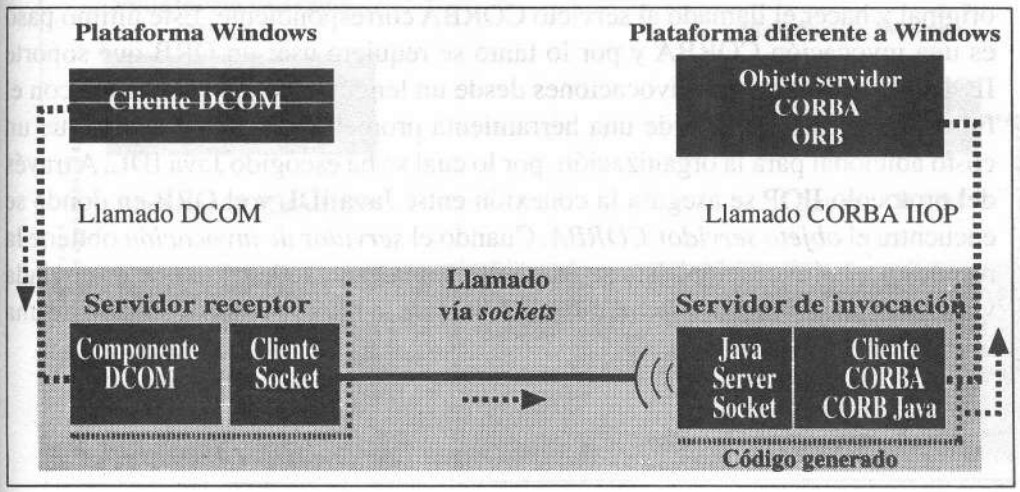
- *Cliente DCOM:* Aplicación cliente que corre sobre una plataforma Windows y realiza llamados a servicios de componentes DCOM. Está construida en cualquier lenguaje de programación que soporta el API DCOM [Chappell, 1996], a saber, Visual C++ 4.2+, Visual Basic 5.0+, Visual J++ o Delphi.

- *Servidor receptor:* Es un componente DCOM encargado de atender llamados de *clientes DCOM* que quieren acceder a servicios CORBA. Cada vez que recibe una solicitud, la convierte a una cadena de texto y la envía a través del mecanismo de *sockets* al módulo *servidor de invocación*, encargado de comunicarse con el *objeto servidor.*

- *Servidor de invocación:* Es un servidor concurrente de *sockets*, que recibe una cadena de texto con los datos necesarios para realizar la invocación remota real y realiza llamados a través del protocolo IIOP (Internet Inter-ORB Protocol) [Siegel, 1996] al *objeto servidor CORBA*. Para este fin, debe examinar la cadena de texto recibida para obtener el nombre del servicio y los valores de los parámetros y realizar una conversión de tipos.

- *Objeto servidor CORBA:* Aplicación servidor que implementa servicios CORBA. Funciona sobre un ORB que soporta el protocolo IIOP.

Figura 1. Esquema de la primera parte del modelo.
Se ilustran los elementos componentes del mecanismo de invocación de servicios CORBA desde un cliente DCOM



A continuación se describe el proceso de invocación remota DCOM- CORBA. El *servidor de invocación*, encargado de realizar los llamados CORBA a los servicios respectivos, debe implementar efectivamente las invocaciones. Esto quiere decir que se deben especificar en tiempo de edición, en su código fuente, los nombres de los procedimientos con los respectivos parámetros que serán invocados en ejecución. Por esta razón, se ha construido una aplicación que se encarga de construir el código fuente del *servidor de invocación* a partir de la interfase en lenguaje IDL [Orfali y Harkey, 1997] del *objeto servidor CORBA*. El código fuente entonces es generado en lenguaje Java y usa el API Java IDL para hacer llamados CORBA vía IIOP al objeto servidor en cuestión. Por lo tanto, el usuario deberá compilar el *servidor de invocación* usando la herramienta JDK (*Java Developers Kit*) versión 1.2, de libre distribución. Una vez compilado este código, debe ser activado y a continuación se debe registrar el *servidor receptor* en el *Registry* [Chappell, 1996] de la máquina cliente. Inicialmente, el cliente crea una instancia del *servidor receptor* e invoca el único servicio que éste implementa, que se ha denominado *Transfer*. El método *Transfer* debe ser llamado con los siguientes parámetros:

- Nombre del *host* donde reside el *servidor de invocación*.
- Nombre del *host* donde reside el *objeto servidor CORBA*.
- Nombre del *objeto servidor CORBA*.
- Nombre del servicio requerido sobre el *objeto servidor CORBA*.
- Un arreglo de parámetros.

Los elementos del arreglo de parámetros se toman como de tipo *Variant* [Chappell, 1996] o su equivalente. La implementación de *Transfer* simplemente empaqueta (es decir, realiza la tarea equivalente al *marshalling* [Farley, 1998]) el nombre del servicio y los parámetros en una cadena de texto que se envía al *servidor de invocación*, quien se encarga de reconvertir los parámetros al tipo de dato original y hacer el llamado al servicio CORBA correspondiente. Este último paso es una invocación CORBA y por lo tanto se requiere usar un ORB que soporte IIOP, que permita hacer invocaciones desde un lenguaje popular estándar (con el fin de evitar la exigencia de una herramienta propietaria) y que no implique un costo adicional para la organización, por lo cual se ha escogido Java IDL. A través del protocolo IIOP se asegura la conexión entre Java IDL y el ORB en donde se encuentra el *objeto servidor CORBA*. Cuando el *servidor de invocación* obtiene la respuesta a la invocación, ésta se convierte a una cadena de texto que es enviada vía *sockets* al *servidor receptor*, quien finalmente la retorna como resultado de una invocación DCOM natural.

2.2. Interconexión CORBA-DCOM

La segunda parte del diseño provee soporte a la invocación de servicios de un componente DCOM desde un cliente CORBA. Es análoga a la primera parte y comprende los mismos elementos descritos anteriormente, pero con diferente funcionalidad. (Figura 2). En este escenario, se identifican los siguientes participantes:

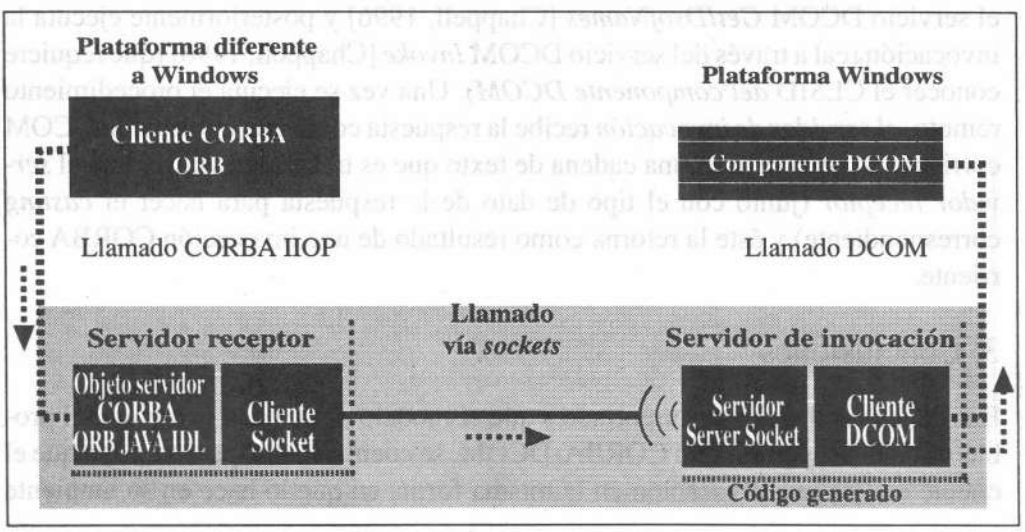
- *Cliente CORBA:* Aplicación que realiza llamados a servicios CORBA y que corre sobre un ORB que soporta IIOP.

- *Servidor receptor:* Es un objeto servidor CORBA que se encarga de recibir solicitudes *de clientes CORBA* que requieren acceder a servicios DCOM. Construye una cadena de texto con los datos necesarios para el llamado remoto y la envía a través de *sockets* al módulo

- *Servidor de invocación:* Servidor concurrente de *sockets* que recibe los datos de la invocación DCOM en una cadena de texto y posteriormente realiza el llamado al servicio respectivo usando el mecanismo *Automation* [Chappell, 1996].

- *Componente DCOM:* Componente de una aplicación servidor DCOM que implementa servicios que pueden ser llamados a través de una interfase *Idispatch.*[Chappell, 1996]

Figura 2. Esquema de la segunda parte del modelo.
Se ilustran los elementos componentes del mecanismo
de invocación de servicios DCOM desde un cliente CORBA.



A continuación se describe el proceso de invocación remota CORBA-DCOM. Inicialmente, el usuario debe activar el *servidor de invocación* y configurar adecuadamente el *Registry* para que pueda hacer llamados al *componente DCOM* que se requiera. El proceso comienza cuando un *cliente CORBA* realiza una invocación a un servicio de un *componente DCOM*, a través de la invocación al método *Transfer* del *servidor receptor*. Esta invocación se hace mediante el protocolo IIOP desde el ORB en donde está situado el *cliente CORBA*, pues el *servidor receptor* es un objeto servidor CORBA que corre sobre un ORB *Java IDL*. En este caso, *Transfer* debe ser invocado con los siguientes parámetros:

- Nombre del *host* donde reside el *servidor de invocación*.
- Nombre del *host* donde reside el *componente DCOM*.
- Identificador de la clase que define el *componente DCOM* (*CLSID* [Chappell, 1996]).
- Nombre del servidor DCOM.
- Nombre del *componente DCOM*.
- Nombre del servicio requerido sobre el *componente DCOM*.
- Un arreglo de parámetros.

El CLSID se obtiene de la interfase del *componente DCOM*. El arreglo de parámetros es de cadenas de texto, de manera que el cliente hace un llamado CORBA corriente. El método *Transfer* hace lo mismo que en la parte anterior del modelo: empaqueta el nombre del servicio y los parámetros en una cadena de texto que se envía al *servidor de invocación* a través de *sockets*. Por su parte, el *servidor de invocación* tiene la tarea de descomponer la cadena de texto para separar los parámetros y convertirlos a sus tipos respectivos con el objeto de hacer el llamado al servicio indicado. Para esto, obtiene la información necesaria usando el servicio DCOM *GetIDsofNames* [Chappell, 1996] y posteriormente ejecuta la invocación real a través del servicio DCOM *Invoke* [Chappell, 1996] (que requiere conocer el CLSID *del componente DCOM*). Una vez se ejecuta el procedimiento remoto, *el servidor de invocación* recibe la respuesta como una invocación DCOM corriente, la convierte en una cadena de texto que es trasmitida vía *sockets* al *servidor receptor* (junto con el tipo de dato de la respuesta para hacer el *casting* correspondiente) y éste la retorna como resultado de una invocación CORBA corriente.

3. Conclusiones

Entre las cualidades más importantes que el modelo aporta a la solución del problema de interoperabilidad CORBA-DCOM, se cuentan la trasparencia con que el cliente realiza una invocación en la misma forma en que lo hace en su ambiente

distribuido y la independencia de soluciones propietarias. Aunque el modelo soporta solamente tipos de datos básicos y no objetos complejos, es un hecho que la mayoría de aplicaciones pertenecen al primer caso. Otra ventaja significativa que conviene resaltar es la cantidad mínima de modificaciones que se deben hacer a las aplicaciones cuando se quiere lograr interoperabilidad bidireccional mediante este modelo, cuya solución es idealmente satisfactoria como transición a un esquema de interconexión que cumpla el nuevo estándar del consorcio OMG (Object Management Group) *CORBA-DCOM Internetworking*.

Referencias

Chappell, D. *Understanding ActiveX and OLE.* Redmond: Microsoft Press, 1996.

Dolgicer, M.; Fischer, P. *DCOM, ActiveX and CORBA must live together.* URL: "http://www.adtmag.com/pub/apr97/fe403.htm", 1997.

Farley, J. *Java Distributed Computing.* O'Reilly, 1998.

Orfali, R. y Harkey, D. *Client/Server Programming with Java and CORBA.* Wiley Computer Publishing, 1997.

Siegel, J. *CORBA Fundamentals and Programming.* New York: John Wiley and sons, 1996.