

Tecnologías para el desarrollo de Sistemas Distribuidos: Java versus Corba

David Basanta Gutiérrez, Lourdes Tajés Martínez

?

Resumen—En los últimos años se está viviendo un auge en el diseño y desarrollo de Sistemas Distribuidos. Más allá de las cuestiones clásicas de diseño de los mismos (acceso y localización transparentes, escalabilidad, tolerancia a fallos, rendimiento, etc) se plantea qué tecnología utilizar para su desarrollo. Se presenta aquí una introducción a los sistemas distribuidos en Java, incluyendo el estándar CORBA del OMG y el sistema propio de Java, RMI, con un componente crítico que permita discernir su aplicabilidad y adecuación al desarrollo de aplicaciones distribuidas.

Palabras Clave—Sistemas Distribuidos, objeto, Java, Corba, ORB, interoperabilidad, integración

I. SISTEMAS DISTRIBUIDOS: ANTECEDENTES

INICIALMENTE, los ordenadores eran grandes centros de computación con entidad propia en centros de investigación o gubernamentales y Universidades. Con el boom de los PC en los 80, todos pudieron tener parte de esa capacidad de cómputo. Los PC, mucho más baratos que minis o mainframes, fueron una gran aportación para empresas y particulares.

Aunque el ordenador personal estaba pensado para ser un elemento de computación autónomo y no formar redes de ordenadores, la posibilidad de *compartir recursos* gracias a la comunicación de varios PC, supone una ventaja que los fabricantes no pudieron ignorar, empezando así la carrera hacia los *sistemas distribuidos*, que tratan de aunar lo mejor de microordenadores y superordenadores a la vez, creando un *ordenador virtual a partir de varios PC*.

A. Orientado a Procedimiento

La comunicación entre dos PC en sistemas UNIX, mejoró mucho cuando BSD introdujo el concepto de *socket*, que permitía que la comunicación entre

procesos sitos en ordenadores distintos no fuera mucho más complicada que la de un programa que supiese leer y escribir ficheros. Los programas que emplean sockets establecen un protocolo de comunicación para poder entenderse.

El RPC, de Sun Microsystems, es el siguiente nivel de abstracción y permite realizar llamadas a procedimientos independientemente de su localización, buscando con ello la transparencia de acceso para el programador.

B. Orientada a Objetos

Las tecnologías Orientadas a Objeto, hacen que estas abstracciones orientadas a procedimiento resulten inadecuadas, y propicia la aparición de sistemas como CORBA, RMI y COM/DCOM.

En este trabajo¹ el término *sistema distribuido* hará referencia al uso de objetos por parte de otros que pueden estar situados o no en la misma máquina, de forma casi transparente[3]. También se hablará de *objetos servidores* que ofrecen servicios, y de *objetos cliente*, que los usan, aunque esta distinción carece de sentido en sistemas distribuidos, donde un objeto puede adoptar simultáneamente ambos roles.

II. JAVA Y LAS REDES

A. SUN

Java es un *entorno de computación*² introducido al público en 1995 por Sun Microsystems³. Considerado como lenguaje de programación, es simplemente un lenguaje cuya sintaxis recuerda la del C++, y tiene,

¹ Soportado en parte por el Proyecto *Interoperabilidad de objetos y componentes mediante el sistema integral orientado a objetos Oviedo*³, Universidad de Oviedo

² Java es el nombre del lenguaje de programación y de la arquitectura. Aquí se le da el nombre de entorno de computación

³ Sun ya era conocida por tecnologías en sistemas operativos (como las que incorpora Solaris) y sistemas distribuidos (como NFS). De hecho, tiene como lema *la red es el ordenador*.

respecto a este, ventajas, que el marketing de la compañía trata de resaltar, e inconvenientes, que trata de ocultar, sin embargo, las librerías de clases y el que todos sus programas se ejecuten en una máquina virtual lo convierten en un lenguaje altamente *portátil*, muy apto para una red con ordenadores y sistemas operativos tan heterogéneos como Internet[1].

B. Máquinas Virtuales

El secreto de la portabilidad binaria de los programas Java reside en las máquinas virtuales. Los compiladores Java no generan binarios para una arquitectura real, sino para una inexistente *máquina virtual Java*, por lo que para usar los binarios, se necesita un emulador encargado de traducir instrucciones del programa a primitivas de la arquitectura anfitriona.

C. Popularidad

La popularidad actual del lenguaje Java es, en parte, fruto de una agresiva campaña de marketing por parte de Sun, además de pactos con la compañía que anteriormente era líder en la fabricación y distribución de navegadores de Internet, Netscape. En todo caso, hoy en día, Java es una plataforma muy popular y casi todos los ordenadores que se fabrican ahora mismo incluyen una máquina virtual capaz de ejecutar sus programas.

III. SISTEMAS DISTRIBUIDOS EN JAVA

A. RMI

RMI[4] fue el primer framework⁴ para crear sistemas distribuidos que apareció para Java. Además, viene integrado en cualquier máquina virtual Java posterior a la 1.0 y está pensado para hacer fácil la creación de sistemas distribuidos a partir de una aplicación cuyas clases ya estén implementadas.

B. CORBA: El estándar de sistemas distribuidos

CORBA es el estándar para la creación de sistemas distribuidos creado por el *Object Management Group* (OMG). Pensado para ser independiente del lenguaje, rápidamente aparecieron implementaciones en las que se podía usar casi cualquier lenguaje.

⁴ Jerarquía de clases que facilitan la programación en el dominio del problema para el que fueron pensadas. Así, AWT es un framework para crear interfaces de usuario de programas en Java.

C. DCOM: La alternativa de Microsoft.

Aunque objeto y componente[2] tienen significado distintos, el nombre utilizado en las tecnologías de Microsoft COM y DCOM, versión distribuida de COM es *componente*. COM/DCOM es un sistema de componentes implementado en todos los sistemas operativos que fabrica Microsoft. La tecnología para crear sistemas distribuidos proporcionada por Microsoft es una versión orientada a componentes del sistema RPC ya comentado. Si bien es verdad que se han hecho algunos esfuerzos para que DCOM aparezca en arquitecturas diferentes a la Win32 (llegando a acuerdos con el gigante del software alemán Software A.G.), no parece que dichos esfuerzos hayan proporcionado resultados visibles y hoy por hoy, DCOM es una tecnología que solo sirve para los sistemas Microsoft.

IV. CORBA.

CORBA, *Common Object Request Broker Architecture*, es una tecnología para crear sistemas distribuidos, creada por un consorcio de fabricantes, agrupados bajo el OMG.

El estándar CORBA define qué ha de incluir una implementación estándar, pero no cómo se han de hacer. Esta tarea se deja de la mano de los diferentes fabricantes. Esta es una de las principales características de CORBA: permite una total libertad a los implementadores siempre que estos respeten unos mínimos orientados a la *interoperabilidad* entre implementaciones.

A. Ventajas

1) Disponibilidad y Versatilidad

Muchas arquitecturas y sistemas operativos cuentan con una implementación de CORBA, lo que hace suponer que se puede usar CORBA en virtualmente cualquier proyecto de sistemas distribuidos.

2) Eficiencia

La libertad de desarrollo ha favorecido la existencia de una pléyade de implementaciones del estándar que se adaptan a multitud de posibles necesidades de los usuarios, generando una competencia que favorece aquellas implementaciones de mayor calidad y con más características.

3) Adaptación a Lenguajes de programación

Además, es posible emplear los servicios de CORBA desde cualquier lenguaje de programación, desde C++, C ó Java, hasta COBOL ó Ada.

B. Inconvenientes

1) Complejidad

Permitir la interoperabilidad de distintos lenguajes, arquitecturas y sistemas operativos hace que sea un estándar bastante complejo, y su uso no sea tan transparente al programador como sería deseable:

1. Hay que usar un compilador que traduce una serie de tipos de datos estándares a los tipos del lenguaje en el que se vaya a programar (IDL)
2. Hay que ser conscientes a la hora de diseñar qué objetos van a ser remotos y cuáles no (los remotos sufren restricciones en cuanto a sus capacidades con respecto a un objeto normal)
3. Es necesario emplear tipos de datos que no son los que proporciona de manera habitual el lenguaje de programación (muchas veces hay que emplear tipos de datos adaptados de IDL).

2) Incompatibilidad entre implementaciones

Muchas empresas ofrecen implementaciones CORBA, si bien el grado de cumplimiento es diverso. Las divergencias entre ORBs radican en detalles que, aunque no hacen imposible aplicar en uno el mismo diseño de un programa pensado para otro, hacen que la adaptación sea fastidiosa. Cuestiones como la colocación de librerías o las diferentes formas de implementar la gestión de la concurrencia, hacen difícil la portabilidad del código y obligan al programador a reciclarse cuando quiere cambiar de ORB. Además, donde el estándar no concreta, las implementaciones pueden variar entre sí, lo que da lugar a molestas incompatibilidades que complican la vida al usuario.

C. Los ORBs.

Los ORBs[6], *Object Request Brokers*, núcleo de cualquier implementación CORBA, transmiten los mensajes que se intercambian cliente y servidor, para lo que se ocupan de:

1. Canalizar las comunicaciones entre los objetos locales y los remotos.
2. Empaquetar los parámetros que el cliente pasa al método remoto y el resultado que el método devuelve al cliente.
3. Localizar al objeto remoto a partir de una referencia.

D. EL IDL.

IDL (*Interface Definition Language*) es un lenguaje de programación pensado exclusivamente para especificar los interfaces de las clases cuyas instancias queremos hacer públicas a objetos remotos que las usaran como clientes.

La necesidad de un IDL viene dada por la *independencia de CORBA respecto a la arquitectura y al lenguaje de programación*. Distintos lenguajes soportan diferentes tipos de datos y tienen distintas formas de especificar clases. Incluso limitándonos a un lenguaje, la ordenación⁵ y el tamaño de un tipo de datos determinado no tiene porqué ser el mismo entre arquitecturas diferentes (por ejemplo, no es lo mismo un entero en un 386 con MS-DOS que en un UltraSparc con Solaris 7).

IDL pone de acuerdo a distintos lenguajes en el formato y tamaño de sus especificaciones. El compilador de IDL transforma una especificación neutral para la plataforma y el lenguaje en otra que puedan entender dicho lenguaje y plataforma.

E. El IIOP: Interoperabilidad entre ORB.

CORBA es neutral respecto al protocolo de red utilizado para comunicar cliente y servidor. Para ello especifica el GIOP (*General Inter ORB Protocol*) que define a muy alto nivel la comunicación entre ORBs diferentes. Para redes de tipo TCP/IP se emplea una instancia de GIOP conocida como IIOP (*Internet Inter ORB Protocol*). Gracias a IIOP, es posible que objetos que emplean ORBs de fabricantes distintos puedan interoperar en redes como Internet.

F. Gestión de la concurrencia.

El comportamiento de un objeto situado en un servidor cuando dos o más clientes quieren hacer uso de sus servicios viene determinado por la política de gestión de la concurrencia que se haya programado en el ORB. CORBA incluye varias políticas que definen cuándo y cómo activa el ORB los objetos en el servidor para atender peticiones.

Desgraciadamente, si se utiliza un mismo proceso para atender todas las peticiones que se hagan, o si se crea uno nuevo para atender cada una de las peticiones es algo de lo que se va a tener que ocupar el programador, aunque algunos ORBs pueden asistir al programador en esa tarea.

G. Servicio de nombrado.

En CORBA hay varias formas de que un objeto situado en una máquina pueda referirse a otro remoto.

⁵ Big-Endian y Little-Endian

1) IOR

Número de gran longitud, que permite identificar de manera única y global a un objeto que ofrece sus servicios en un entorno distribuido. Lo genera automáticamente el ORB de forma que no pueda haber dos objetos con el mismo identificador por muy grande que sea la red. Usa para ello datos aleatorios, y otros escogidos a partir del ordenador sobre el que se ejecuta, la implementación del ORB, etc.

2) Asignación de Nombres

Dándole previamente un nombre al objeto, otro que quiera usar sus servicios podría emplear una notación tipo URL como `iiop://nombre_host:puerto/nombre_objeto`⁶. Así, si en `máquina.inf.uniovi.es`, existe el objeto `dns`, cualquiera que quiera acceder a `dns` sólo tiene que solicitar a su ORB una referencia a `iiop://máquina.inf.uniovi.es/dns`. Esta forma de nombrado es más fácil de recordar para la mayor parte de los seres humanos, aunque seremos nosotros los que tendremos que asegurarnos de su unicidad (aunque solo dentro de los límites de la máquina en la que se registre).

Ambos sistemas de nombrado tienen el inconveniente de *no ser transparentes en cuanto a la localización* ya que, si movemos los objetos servidores, tendremos que adecuar a los objetos clientes para que los busquen en otro lado.

H. Servicios adicionales de CORBA.

Las implementaciones CORBA pueden ofrecer servicios adicionales voluntariamente (aunque no es necesario para ser certificado *CORBA compliant* por el OMG).

Un ejemplo de estas facilidades es el sistema de *suscripción de eventos*, que permite que un objeto se suscriba a eventos generados por otro. El propósito de este servicio es el de mejorar la eficiencia disminuyendo el tráfico de la red. Por ejemplo, si hay varios objetos clientes esperando a que suceda algo en el objeto que presta servicio en el servidor, en vez de hacer *polling*, podrían solicitarle a este que les envíe una notificación cuando eso ocurra.

I. Seguridad.

El estándar CORBA no se preocupa de la seguridad implementada en el sistema distribuido. Si por alguna razón se requiere restringir el uso de los recursos controlados por un determinado objeto, debe hacerlo el usuario.

J. Integración de CORBA en Java.

CORBA, como especificación, es absolutamente independiente de la plataforma o el lenguaje, por lo que no han tardado en aparecer implementaciones de CORBA con soporte de Java (como *VisiBroker* ó como *ORBacus* por ejemplo).

Esto se traduce principalmente en un traductor IDL ? tipos básicos de Java y en un framework, que permiten acceder a la librería de servicios del ORB. Algunas de estas librerías son nativas, en vez de emplear implementaciones 100% Java para soportar Java, lo que puede dar algunos problemas de portabilidad o imposibilitar la ejecución de aplicaciones dentro de los navegadores con soporte para Java.

Pero la integración Java-CORBA incluirá interesantes mejoras cuando OMG introduzca la nueva versión del estándar, CORBA 3.0[9]. De hecho, parte de la mejora en esta integración es justo lo inverso de lo que había hasta ahora: una correspondencia *java-idl*. Al contrario que el compilador de IDL a Java, que convierte una especificación en formato neutral en clases que puede usar un compilador java, el traductor *java-idl* extraerá la interfaz de las clases java y las traducirá a una especificación IDL, por lo que será más fácil emplear clases java por programas que usen CORBA y que estén implementados en otros lenguajes.

K. Limitaciones e Inconvenientes CORBA.

1. El sistema no es transparente al programador. Las diferencias para el programador que quiera usar un determinado objeto con respecto a las de emplear uno local, se reducen a la inicialización del mismo. En vez de inicializarlo normalmente, hay que pedir al ORB (via IOR o usando un nombre más inteligible), una referencia al objeto remoto y luego convertirlo al tipo de objeto a manejar.
2. Los objetos remotos se pueden usar por referencia, pero no por valor. Así, cuando se haga uso de los métodos de un objeto remoto (al que se accede por referencia), solo se le pueden pasar como parámetros (y el método solo podrá devolver como resultado) tipos de datos contemplados en el IDL. Afortunadamente, este problema queda resuelto con CORBA 3, que sí soporta el paso de parámetros por valor.
3. Múltiples implementaciones de CORBA dan lugar a múltiples incompatibilidades. El estándar CORBA define a alto nivel qué funciones debe proporcionar un ORB y cómo han de interoperar

⁶ Para que este tipo de URL pudiera ser comprensible para el ORB, debería implementar el protocolo IIOP

estos entre sí, lo que garantiza cierto grado de compatibilidad, pero el cómo se ofrezca esa funcionalidad al programador es algo que está al libre albedrío del fabricante del ORB. Es más, parte de la funcionalidad del estándar CORBA no es de obligado cumplimiento por parte de las compañías fabricantes para poder anunciarse como CORBA-compliant. En estas condiciones es muy difícil pensar que un programa que haya sido programado pensando en un ORB concreto, pueda funcionar bien con una simple recompilación.

4. El estándar CORBA está poco preparado para usarse en entornos embebidos (electrónica de consumo, asistentes digitales) o que requieran soporte de tiempo real. Para el primer caso, se diseñó en CORBA 3 un subconjunto llamado *Minimum CORBA* que, al ser más ligero, encaja mejor en los sistemas embebidos, donde el control del consumo de recursos es vital. Para solucionar el segundo problema, CORBA 3 introducirá *Real-Time CORBA*, que introducirá modelos de prioridad para conseguir un comportamiento predecible de los programas que lo usen.

V. RMI.

RMI (*Remote Method Invocation*) es parte de la librería de clases de Java que permite la creación de sistemas distribuidos en Java.

A. Características particulares de RMI.

Al contrario que otras tecnologías de sistemas distribuidos, RMI no busca la colaboración de objetos de distintas plataformas, programados en diferentes lenguajes, Java se encarga de solucionar los problemas de heterogeneidad. Así, su API es más sencillo y natural al no contemplar que tipos de datos (y sus tamaños) existan o no en los lenguajes en los que se implementan cliente y servidor.

B. Gestión de la concurrencia.

La gestión de la concurrencia en RMI, como muchas de las cosas que lo diferencian de CORBA es extraordinariamente sencilla e inflexible. Para cada cliente que trate de acceder a un objeto remoto, el servidor creará un nuevo hilo que se encargará de darle servicio. Si varios hilos del mismo cliente realizan distintas peticiones al servidor, compartirán un mismo hilo en el servidor.

C. Servicio de nombrado.

El nombrado de objetos es sencillo pero dependiente del ordenador donde resida el objeto servidor en cada instante. Aún así, está pensado de tal forma que difícilmente su uso creará problemas al usuario, ya que emplea la notación URL del tipo `mi://nombre_host:puerto/nombre_objeto`.

Un programa que acompaña a las implementaciones Java es el RMIRegistry. El registro rmi (`rmiregistry`) comunica objetos clientes con servidores. Conceptualmente, cuando un cliente quiere obtener una referencia a un objeto remoto, pregunta por él al `rmiregistry` donde resida dicho objeto. El registro lleva la cuenta de los objetos exportados⁷ que residen en esa máquina (ya que todos ellos han sido dados de alta en el registro por el programa que se haya encargado de crearlos e inicializarlos), por lo que comprueba que la solicitud del cliente puede ser atendida, y caso de ser así, devuelve la referencia esperada.

El RMIRegistry puede ser ejecutado como servicio de Windows NT o como demonio en UNIX (y similares).

D. Paso de parámetros por valor y serialización.

RMI soporta que objetos clientes puedan emplear objetos remotos por valor y por referencia. El uso de un objeto remoto por referencia no es nada nuevo, cualquier objeto exportado ante el RMIRegistry se pasa automáticamente por referencia a cualquier cliente que quiera usarlo.

La novedad es el paso de objetos por valor, para lo que se emplea la librería de serialización[5] del API de Java. Para pasar un objeto por valor del servidor el cliente, debe implementar una clase abstracta que les obliga a definir cómo almacenar en un flujo de datos (como por ejemplo un fichero) los datos importantes del objeto serializable y cómo reconstruirlo a partir de esos mismos datos. Muchas de las clases del API de Java ya son serializables por lo que otras clases que las usen (por herencia o agregación) no tienen que ocuparse de serializarlas.

Cuando se necesita llevar un objeto serializable de una máquina a otra (porque sea un parámetro de un método de un objeto remoto que empleamos por referencia, o porque sea el resultado que devuelve dicho método al cliente), se serializa, se lleva el flujo de una máquina a la otra, y una vez en el ordenador huésped, se reconstruye y se usa.

⁷ Esto es, los que ofrecen sus servicios a otros objetos.

La ventaja del paso por valor en un sistema distribuido radica en que se minimiza el tráfico en la red ya que, una vez recibido el objeto, todas las comunicaciones con él serán locales.

E. Seguridad.

Por defecto, RMI no incluye ninguna facilidad para restringir el uso de las clases desde clientes no autorizados, aunque el usuario siempre puede suplir parte de esta funcionalidad usando los servicios del sistema operativo, o haciendo ese trabajo él⁸ mismo. Aún así, no se puede confiar en la seguridad de un sistema solo porque se haya autenticado al usuario al inicio de la conexión si es que el canal de comunicación no es seguro. La plataforma Java 2 incluye el soporte de comunicaciones seguras en RMI usando el estándar SSL (*Secure Sockets Layer*).

VI. INTEROPERABILIDAD DE TECNOLOGÍAS

Afortunadamente para los usuarios que necesitan crear un sistema distribuido, existen modos de poder utilizar un objeto cliente implementado según un estándar de sistemas distribuidos, desde otro diferente. Un ejemplo es la integración entre COM y CORBA desarrollada por IONA Technologies[7].

Asimismo, OMG y Sun están trabajando para hacer que CORBA y RMI sean interoperables de manera transparente al programador y al usuario. Además, la plataforma Java 2 viene con soporte de RMI y de CORBA⁹ simultáneamente.

El rendimiento es menor, lo que se agrava especialmente si la comunicación entre dos objetos implementados en estándares distintos es intensa. Además, la interoperabilidad suele suponer tener que conformarse con el mínimo común denominador de las facilidades que incorporen los sistemas originales.

VII. CONCLUSIONES.

¿Cual de las tres tecnologías para crear sistemas distribuidos de las aquí descritas es la apropiada? No existe una respuesta clara.

DCOM está especialmente pensado para poder utilizar componentes de Microsoft. Si en la aplicación se pretende acceder de manera remota a un componente de la suite Office97 o al Internet Explorer, o si se está utilizando como herramienta de programación Microsoft Visual J++, entonces COM/DCOM es

⁸ En UNIX, por ejemplo, esto se configura habitualmente, en los ficheros /etc/hosts.allow y /etc/hosts.deny

⁹ Esto es, incluye un ORB

posiblemente la única solución. El problema es que al ser DCOM una tecnología centrada casi en exclusiva en los sistemas win32, hay que prescindir de una de las grandes ventajas que permiten los sistemas distribuidos que es la interoperabilidad de entornos heterogéneos.

Por otra parte, si tanto cliente como servidor van a usar Java, se quiere una solución sencilla para crear un sistema distribuido y valen las opciones por defecto de los diseñadores de RMI, esta es la mejor elección. RMI es sin duda la opción más integrada en Java, la de uso más natural para un programador de Java y que además aporta un buen número de utilidades. Sin embargo su sencillez se puede transformar en inflexibilidad si se quieren hacer cosas que los creadores de RMI no contemplan.

Por último, si se quiere garantizar una casi absoluta capacidad de elección para cualquier parámetro del sistema distribuido o si no vale Java para ambos extremos de la comunicación, la solución es CORBA.

Escoger CORBA sería sólo la primera de las decisiones que habría que tomar para crear un sistema distribuido, la siguiente sería escoger qué implementación de CORBA escoger, para lo cual hay muchos factores que evaluar: soporte del lenguaje y plataforma que se quiere usar, precio, cumplimiento del estándar, servicios adicionales, velocidad o soporte son sólo algunas de las posibilidades. La elección de la implementación es algo que no se debe menospreciar porque una vez tomada una decisión es difícil cambiar. Esto es así puesto que aunque las diferentes implementaciones son interoperables, existen bastantes diferencias a la hora de programar como para que exista portabilidad binaria o de fuentes, incluso si ambas implementaciones soportan el mismo subconjunto o superconjunto de CORBA.

Por otro lado, CORBA es casi lo opuesto a RMI. Donde este aporta sencillez, CORBA aporta un cierto nivel de complejidad. El premio a este esfuerzo es un sistema es flexible y que se adapta a casi cualquier entorno de computación existente.

VIII. REFERENCIAS

- [1] IEEE Software. Software Components. Noviembre-Diciembre, 1998.
- [2] David Basanta Gutiérrez. Desarrollo de un módulo para clientes web bajo Java. PFC EUITIG, Septiembre 1997
- [3] Visigenic. Distributed Object Computing in the Internet Age. Febrero 1997.
- [4] Sun Microsystems. RMI Documentation Beta Draft. Diciembre 1996.
- [5] Sun Microsystems. Serialization Doc. Beta Draft. Diciembre 1996.
- [6] Sai-Lai Lo, The omniORB2 user's guide. Olivetti & Oracle Research Lab.1997.
- [7] Ronan Geraghty et al.COM-CORBA Interoperability. Prentice Hall, 1997

- [8] Using distributed objects to build the Stanford digital library Infobus. IEEE Computer Magazine. Febrero 1999.
- [9] Jon Siegel. A Preview of CORBA 3. IEEE Computer Magazine. Mayo 1999.