DISEÑO DE SISTEMAS DE INFORMACIÓN DISTRIBUIDOS. ESPECIFICACIONES 'CORBA'.

Adolfo López Paredes
ETSII Valladolid, adlo@eis.uva.es

Juan José de Benito Martín
ETSII Valladolid, debenito@eis.uva.es

Ricardo del Olmo Martínez
ETSII Valladolid, olmo@eis.uva.es

Resumen

El siglo XXI ya ha sido denominado como el de la Sociedad del Conocimiento y de la Información. En este marco, las capacidades y la actividad que va a desempeñar el Ingeniero Industrial van a estar muy próximas a los Sistemas de Información [SI], las aplicaciones de Inteligencia Artificial [IA] y la Inteligencia Artificial Distribuida [DAI].

En este artículo acotamos las claves del éxito para construir sistemas de información distribuidos. En ellos se conjugan las aplicaciones autónomas, inteligentes y distribuidas, con las aplicaciones existentes.

El empleo de estándares como CORBA facilita esta tarea. Demostramos además que la generalización del uso de la especificación CORBA permitirá consolidar los sistemas virtuales de valor y redefinir la gestión de la cadena de suministro.

Palabras Clave: Sistema de Información, Sistemas de Información Distribuidos, CORBA.

1 SISTEMAS DISTRIBUIDOS

En la década de los ochenta se empezaron a desarrollar microprocesadores más poderosos, de 32 e incluso de 64 bits. Se extendió el uso de los ordenadores, y con ellos las bases de datos, los sistemas de soporte a la decisión, los métodos de optimización, la simulación, y otras herramientas computacionales.

Este es el origen de los primeros sistemas de información empresarial: previsiones comerciales, gestión de almacenes, análisis contable, planificación financiera, etc.

El salto cualitativo en el desarrollo de aplicaciones y en la evolución de los sistemas de información fue el uso de redes de área local (LAN) de alta velocidad y la incorporación de los modelos Cliente/Servidor.

De hecho, la utilización de los sistemas MRP y la evolución hasta los ERP se hace posible gracias a la extensión y adopción generalizada de la arquitectura Cliente/Servidor.

Mediante estas tecnologías se podían conectar docenas, e incluso cientos de máquinas, permitiendo la transferencia de información y la colaboración entre ellas. Por ello hoy en día no sólo es posible, sino además fácil, crear sistemas de cómputo compuestos por un gran número de CPU's (figura 1).

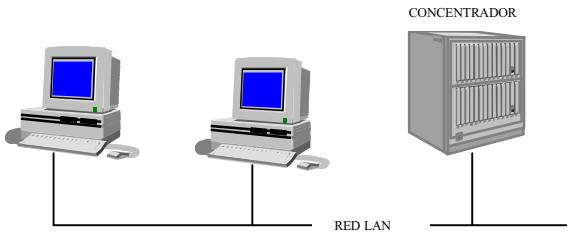


Figura 1. Sistema distribuido unido mediante una red LAN.

Estos sistemas reciben el nombre genérico de **sistemas distribuidos**, en contraste con los **sistemas centralizados**, que constan de una única CPU, sus periféricos y terminales.

1.1 ARQUITECTURA DE LOS SISTEMAS DISTRIBUIDOS

El desarrollo de sistemas distribuidos se fundamenta en la idea de que un grupo de ordenadores personales conectados entre sí y a algunos pequeños servidores, proporcionan mayor rendimiento y mejor relación "calidad/precio" que las grandes computadoras.

Estas características se sitúan entre las principales ventajas que se persiguen en el diseño de sistemas distribuidos, además de algunas otras entre las que podemos resaltar su capacidad para la expansión, la protección contra fallos y la duplicación de la información.

Sin embargo estos sistemas también poseen algunos inconvenientes. Mencionar entre ellos la dependencia del funcionamiento de la red (se está trabajando en el diseño de redes tolerantes a fallos) y problemas de seguridad (estos sistemas se construyen normalmente sobre una filosofía de sistemas abiertos).

Para resaltar las ventajas y eliminar en lo posible los inconvenientes es necesario establecer unos criterios de diseño que se deben cumplir en la construcción de este tipo de sistemas:

- *Transparencia:* el diseñador debe crear el sistema de forma que el usuario no perciba que la aplicación se está ejecutando en un conjunto de máquinas, sino que piense que se ejecuta en una sola.
- *Flexibilidad:* debe permitirse la modificación de la configuración física del sistema cuando se necesite, para así adaptarse a las necesidades de cada momento.
- *Confiabilidad:* la fiabilidad de un sistema distribuido (probabilidad de que el sistema no falle), debe ser mayor que la de un sistema con un único procesador. Si alguna máquina falla, alguna otra debe encargarse del trabajo que tuviera encomendado.
- *Desempeño:* el sistema debe realizar la tarea para la que fue diseñado de una forma adecuada y en el menor tiempo posible. El desempeño de un sistema distribuido estará supeditado a la velocidad de transmisión de datos de la red.
- *Escalabilidad:* que permite que el sistema y las aplicaciones se expandan en forma escalar sin cambiar la estructura propia del sistema o los algoritmos de aplicación.

1.2 MODELO CLIENTE/SERVIDOR

Un sistema distribuido puede tener dos formas de comunicarse, utilizando el modelo de protocolos por capas (también denominado modelo OSI) o mediante el modelo Cliente/Servidor.

El modelo de protocolos por capas, establece siete niveles llamados capas, cada uno de los cuales se encarga de una tarea determinada. La red o capa física se encarga de la transmisión de los ceros y unos, la capa de enlace de datos se encarga de la detección y corrección de los errores que se producen en la capa física, etc.

Para que los equipos se pueda comunicar mediante este modelo, es necesario que haya puntos de acuerdo en todos los niveles (desde los detalles de bajo nivel, de transmisión de los bits; hasta los detalles de alto nivel de la forma en la que se debe expresar la información).

Para conseguir esto se va añadiendo información al mensaje en cada uno de estos niveles, que indica cómo se va a establecer la comunicación en cada uno de ellos (figura 2).

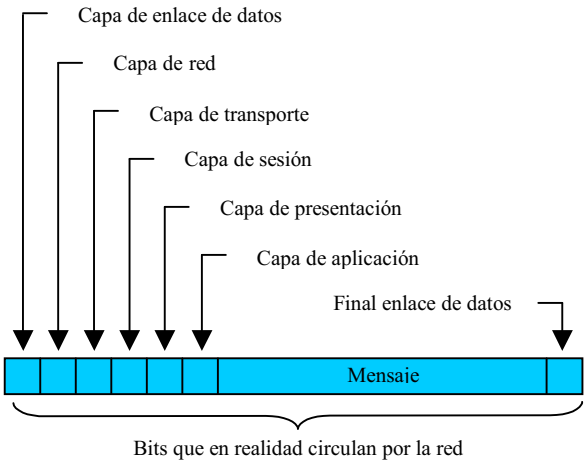


Figura 2. Esquema de un mensaje mediante protocolo por capas.

Cuando el mensaje llega a su destino todos los encabezados deben ser decodificados. Por lo tanto, es necesaria una codificación y una decodificación que consume gran cantidad de recursos de la red.

Mediante el modelo Cliente/Servidor la aplicación debe diseñarse para que funcione mediante dos procesos lógicos distintos: el cliente y el servidor. A pesar de que cliente y servidor pueden coexistir en un mismo ordenador, la mayoría de las veces se ejecutan desde distintos equipos.

Normalmente un cliente es un ordenador personal que ejecuta algún tipo de software de usuario final. Este software, puede enviar una consulta al servidor y procesar la información que le llega de éste. El servidor recibe y procesa la consulta de parte del cliente y devuelve la información solicitada.

Cada proceso, cliente y servidor, funcionan de forma independiente, realizando tareas específicas para cada aplicación. La unión de cliente y el servidor se realiza a través de una red que permite el intercambio de la información necesaria para el desarrollo de la aplicación. En la figura 3 se puede ver un esquema del funcionamiento del modelo Cliente/Servidor.

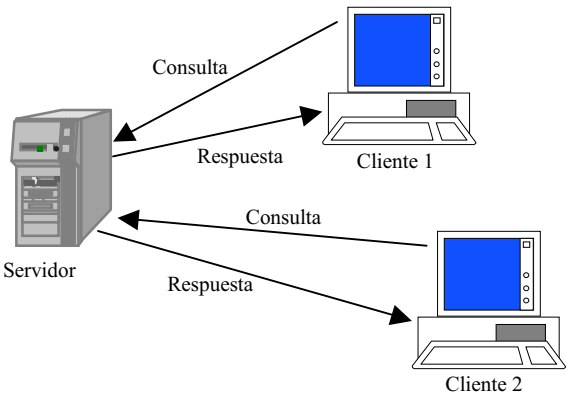


Figura 3. Funcionamiento del modelo Cliente/Servidor.

Dentro del modelo Cliente/Servidor existen diversas variantes que dependen de donde se ejecuten los distintos elementos de la aplicación: administración de los datos, lógica de la aplicación y lógica de la presentación. Así podemos tener:

- *Presentación distribuida*. El cliente sólo asume parte de las funciones de presentación de la aplicación.
- *Presentación remota*. En esta variante, el cliente se encarga de toda la lógica de la presentación.
- *Proceso distribuido o cooperativo.* En el cliente se lleva a cabo, además de la presentación, parte de la lógica de aplicación.
- *Gestión de datos remota*. En este enfoque, el cliente realiza tanto las funciones de presentación como los procesos.
- *Bases de datos distribuidas*. En esta variante, el reparto de tareas es como en el anterior y además el gestor de base de datos divide sus componentes entre el cliente y el servidor.

1.3 COMPONENTES DE UN SISTEMA CLIENTE/SERVIDOR

La aplicación cliente acepta información del usuario, con esa información realiza una consulta al servidor, el cual, después de procesar la información, envía la respuesta al cliente. Por último, el cliente procesa la respuesta y la muestra al usuario de una forma atractiva y adecuada. Por lo tanto, los elementos básicos en cualquier aplicación Cliente/Servidor son el cliente, las conexiones de red y el servidor.

1.3.1 EL CLIENTE

El cliente es cualquier aplicación de software desarrollada para tal finalidad. También puede ser cliente un procesador de texto, una hoja de cálculo, etc.

Otra forma de definir un cliente, es cualquier tipo de aparato que se comunica con un servidor a través de una red. Esto significa que un cliente puede ser un PC, una estación de trabajo, un ordenador portátil, etc.

En resumen, podemos decir que las funciones que generalmente realiza un cliente son:

- Manejo de la interface de usuario.
- Captura y validación de los datos de entrada.
- Generación de consultas e informes sobre las bases de datos.

A la hora de diseñar un sistema, el diseñador debe tener en cuenta algunos aspectos particulares del cliente, algunos de tipo físico y otros de tipo aplicación. Los principales son:

- Procesador.
- Aplicación de desarrollo.
- GUI.
- Middleware.
- Sistema operativo.

Todos los elementos anteriores tienen su relevancia en el desarrollo de una aplicación Cliente/Servidor, sin embargo, por su novedad y por el tema que nos ocupa, una de las más relevantes en este tipo de modelos es el middleware.

1.3.2 LA RED

La red, en el modelo Cliente/Servidor, proporciona un mecanismo de transporte desde el cliente al servidor y viceversa. Una red es realmente una colección de tarjetas de red, concentradores, routers y cables unidos todos juntos para formar un grupo de ordenadores conectados.

La comunicación a través de la red se realiza mediante protocolos, métodos específicos para transmitir información a lo largo de la red. Los protocolos más utilizados son TCP/IP, IPX/SPX y NetBIOS.

Para que el cliente y el servidor puedan comunicarse es necesario que ambos trabajen con el mismo protocolo. Los servidores pueden configurarse para que funcionen con varios protocolos, por los que pueden adecuarse al lenguaje que utilice cada cliente.

1.3.3 EL SERVIDOR

Se puede definir un servidor como una máquina o proceso que proporciona un servicio a otro proceso. Los servidores suelen ser ordenadores que existen en una red y que están configurados para proporcionar

recursos como, archivos, datos, páginas web, procesado de aplicaciones, etc.

Las funciones que realiza un servidor son:

- Gestión de periféricos compartidos.
- Control de accesos concurrentes a las bases de datos compartidas.
- Enlaces de comunicaciones con otras redes de área local o extensa.
- Servicio a peticiones de clientes.

1.4 MIDDLEWARE

El middleware se puede definir como una capa de software intermedio que nos permite aislar la aplicación de las complejidades del sistema operativo y de las conexiones de red (figura 4).

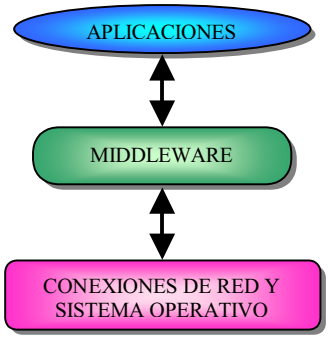


Figura 4. Esquema del funcionamiento del middleware.

Los primeros desarrollos de sistemas cliente/servidor, exigían a los desarrolladores que interaccionasen con los protocolos de red para acceder a la información, o a un servicio que existía en otro ordenador. De esta manera los desarrolladores estaban sujetos a un único protocolo con lo que quedaba limitada su compatibilidad con otras plataformas y redes.

Para evitar este problema, apareció el middleware, ya que sus componentes de red funcionan como un convertidor de protocolos. La conversión a diferentes protocolos ocurre sin que la parte de la aplicación que reside en el cliente o en el servidor perciba.

Además de la traducción de protocolos, middleware se encarga de traducir los datos cuando estos pasan de un ordenador a otro. Por ejemplo los datos que residen en una mainframe son muy diferentes a los que residen en un entorno Windows o UNIX.

Por último, se puede decir que fue la llegada del middleware avanzado lo que realmente permitió el desarrollo de los sistemas Cliente/Servidor, ya que gracias a él se pudieron unir el cliente y el servidor en una arquitectura de dos capas, y se pudieron mantener unidos muchos sistemas diferentes para formar un único sistema virtual.

1.4.1 TIPOS DE MIDDLEWARE

Hay que decir que cualquier intento de clasificación de los middleware es una tarea complicada, ya que hay muchos tipos y, al igual que las tecnologías de la información, cambian muy rápidamente. Como primera aproximación podríamos clasificarlos en:

- *Middleware tradicional*. Este tipo de middleware es el que permite que diferentes sistemas se mantengan unidos para formar un único sistema virtual. Utilizando estos middleware se puede acceder a cientos de computadoras y servidores a través de una única aplicación, que deberá utilizar un conjunto de APIs. Los tipos de middleware tradicional son: llamadas a procedimientos remotos (RPC), middleware orientado a mensajes (MOM), object request broker (ORB), ambiente de computación distribuida (DCE), TP Monitors y middleware de acceso a mainframes.
- *Middleware orientado a base de datos*. Este tipo se refiere al middleware construido para el acceso específico a bases de datos. Este middleware puede utilizar otro middleware tradicional, como puede ser un RPC, para mover información desde y hacia la base de datos. Algunos tipos de estos Middleware son: Struture Query Languaje (SQL) y Microsoft´s Open Database Connectivity (ODBC).

1.5 ORB

Los tipos de middleware tradicionales mencionados anteriormente presentan determinadas características que los hacen particularmente apropiados para diversas funciones.

Los middleware que son objeto de nuestro estudio son los ORB, ya que es un mecanismo que permite la comunicación entre diferentes objetos que están situados local o remotamente, utilizando una interfaz común y un protocolo determinado, TCP/IP.

Un ORB puede hacer consultas a otro ORB, así como procesar las consultas hechas desde otros. La idea es garantizar la interoperabilidad de todos los objetos aunque estén situados en distintos ordenadores conectados entre sí.

Mediante estos middleware se puede hacer que diferentes objetos, escritos en diferentes lenguajes, y soportados por diferentes arquitecturas, puedan comunicarse entre sí.

La figura 5, muestra un esquema de la comunicación entre diferentes objetos a través del ORB.

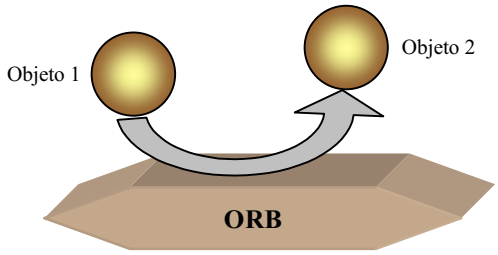


Figura 5. Comunicación entre objetos a través de ORB.

Existen dos corrientes que tratan sobre el intercambio de información entre los distintos objetos:

- *CORBA* (Common Object Request Broker Architecture) es el estándar. Mediante CORBA se pueden crear interfaces de comunicación entre los distintos objetos utilizando para ello el IDL (Interface Definition Language).
- *DCOM* (Distributed Component Object Model) ha sido desarrollado por Microsoft. Utiliza el modelo COM (Component Object Model) y genera las interfaces mediante el ODL (Object Definition Language).

2 SISTEMAS DE INFORMACIÓN DISTRIBUIDOS

Los sistemas de información distribuidos se caracterizan por utilizar la programación orientada a objetos y las capacidades que brindan los ORB, para generalizar el modelo Cliente/Servidor consiguiendo más flexibilidad y mejores prestaciones.

Podríamos definir los sistemas de información distribuidos como redes de objetos, localizadas en unidades de procesamiento autónomas, que se comunican entre si para constituir aplicaciones completas (figura 6).

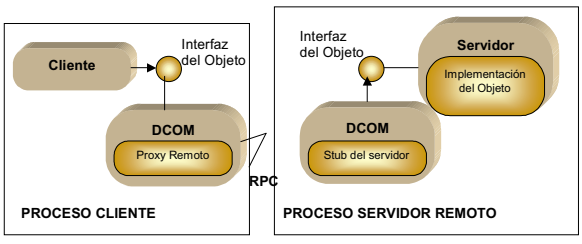


Figura 6. Sistema de información distribuido.

Cada uno de los objetos puede usar cualquier servicio proporcionado por otro objeto en el sistema o incluso en sistemas diferentes.

En estos sistemas no existe una distinción clara entre las aplicaciones cliente y servidor, ya que los objetos que actúan como clientes también pueden actuar como servidores cuando sus métodos son invocados por otro objeto.

Cuando los objetos tienen capacidades autónomas de decisión reciben el nombre de *'agentes'* [1]. Esto ha sido producto de la interacción entre la IA y la Informática Distribuida.

La industria de agentes es una de las más avanzadas tecnológicamente y permite el desarrollo de los primeros: *"sistemas de información inteligentes"*; destinados a ocupar un papel fundamental en la redefinición de los sistemas de valor y los sistemas de información-comunicación entre empresas.

Las tecnologías actualmente más empleadas para la comunicación de objetos/agentes distribuidos son las ya mencionadas CORBA y DCOM, y RMI (Java Remote Method Invocation).

Mientras que CORBA fija un marco para el desarrollo de sistemas distribuidos basado en la definición de interfaces, que otras empresas se encargan de materializar, DCOM y RMI definen tanto la especificación como la tecnología para integrar componentes.

3 LA ESPECIFICACIÓN CORBA

CORBA es una especificación normativa desarrollada por la Object Management Group (OMG) para proporcionar un marco de desarrollo de aplicaciones distribuidas basadas en objetos. Esta especificación se traduce en una arquitectura, OMA (Object Management Architecture), que permite la comunicación de objetos a través de una red, con independencia del sistema operativo, lenguaje de programación y plataforma en la que se desarrollen los objetos.

Para conseguir esto hay que definir una interfaz estándar que es la que permite la comunicación entre los objetos, y que se define utilizando el lenguaje de definición de interfaces IDL (Interface Definition Languaje).

Un elemento clave dentro de esta estructura es el Object Request Broker (ORB) que establece relaciones Cliente/Servidor entre objetos. Mediante la utilización de un ORB, un cliente puede invocar transparentemente un método de un objeto servidor que se encuentre en la misma máquina o en otra distinta. El ORB intercepta la llamada realizada por el objeto cliente, pasa los parámetros, invoca el método en el objeto servidor y retorna los resultados.

Un aspecto característico de esta estructura es que no permite el paso de objetos por valor, por lo que las comunicaciones entre los objetos se hacen mediante referencias a los objetos.

El proceso de comunicación se ha mostrado en cinco pasos en la figura 7: el cliente hace saber al ORB que necesita un servicio de un objeto; el ORB se comunica con el objeto que devuelve una referencia al objeto; con ella el cliente invoca al método deseado que se ejecuta en el objeto; y una vez obtenidos los resultados, se devuelven al cliente.

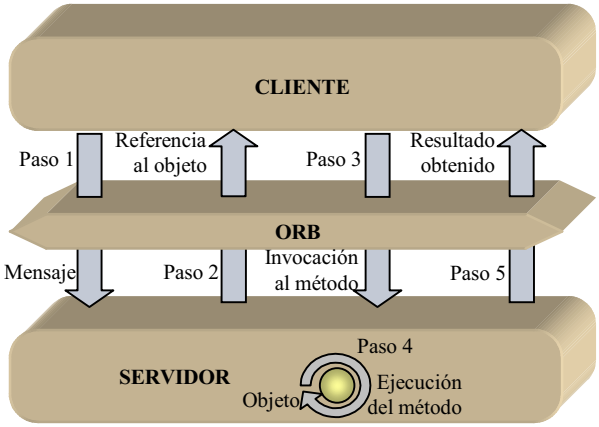


Figura 7. Comunicación del cliente con la implementación del objeto a través de ORB.

Una característica muy importante de CORBA y que ha quedado plasmada en la figura 7 es que todas estas operaciones se hacen a través del ORB, sin que haya comunicación directa entre el cliente y la implementación del objeto.

Finalmente resaltar que mediante esta comunicación no es necesario que el cliente sepa dónde se localiza el objeto que ejecutará el método, el lenguaje en el que está programado, el sistema operativo, ni cualquier otro aspecto que no sea su interfaz.

3.1 ARQUITECTURA CORBA

La arquitectura OMA está constituida por los siguientes elementos (figura 8):

- Los objetos de aplicación.
- El ORB.
- Los servicios CORBA.
- Las facilidades CORBA.
- Los dominios CORBA.

Dentro de esta arquitectura desempeñan un papel preponderante el ORB, ya que la estructura de CORBA y su funcionamiento giran en torno a él.

La estructura de un sistema distribuido basado en CORBA es la que se muestra en la figura 8. En ella pueden apreciarse dos partes: cliente y servidor.

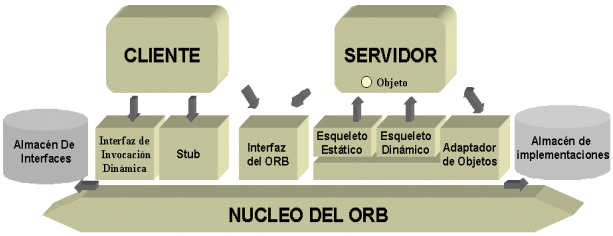


Figura 8. Elementos que intervienen en CORBA.

Se entiende por cliente, el objeto o aplicación que hace una petición a otra. La parte cliente está formada por:

- *Los Stubs del Cliente*. Es el código que se genera cuando se compila la interfaz y proporciona una capa intermedia entre el cliente y el núcleo del ORB.
- *La Interfaz de Invocación Dinámica*. Esta interface se utiliza para el caso de que cliente desee obtener información sobre la interfaz IDL de un objeto en tiempo de ejecución.
- *El almacén de interfaces*. Este componente del ORB, consiste en una base de datos modificable en tiempo de ejecución, donde se almacenan las interfaces de los objetos que manipula el ORB.
- *La interfaz del ORB*. Consiste en unas pocas librerías de servicios locales para realizar labores auxiliares en la aplicación.

En el lado del servidor el ORB localiza al Adaptador de Objetos del objeto y le transmite un mensaje que contiene el nombre del servicio a invocar y sus parámetros. La implementación recibe a través del esqueleto del objeto los datos necesarios, ejecuta el servicio y retorna el resultado para que sea enviado al cliente en forma de un nuevo mensaje.

Los elementos que componen la parte servidora son los siguientes:

- *El esqueleto estático del servidor*. Son los equivalentes al stub del cliente y son creados por el compilador IDL para hacer un puente entre el ORB y la implementación del objeto.
- *El esqueleto de interfaces dinámicas (DSI)*. Proporciona un mecanismo de enlazado en tiempo de ejecución para servidores que necesitan manejar llamadas a métodos que no tienen esqueletos estáticos definidos.
- *El Adaptador de Objetos*. Proporciona una interfaz entre el ORB y el objeto, permitiéndole al ORB preparar el objeto para recibir solicitudes, y al objeto comunicarle al ORB que se encuentra listo para recibir solicitudes.

- *El almacén de implementaciones*. Proporciona en tiempo de ejecución un almacén de información acerca de las clases que el servidor soporta, los objetos instanciados y sus identificadores, es decir, nos da una idea de los tipos de objetos que pueden ser definidos en ese servidor. Es el lugar donde busca el ORB, cuando ha recibido una petición del cliente, para saber exactamente la localización del objeto que debe recibir la petición.
- *La interfaz del ORB*. Consiste en unas pocas APIs de servicios locales iguales a las de la parte cliente.

3.2 LENGUAJE DE DEFINICIÓN DE INTERFACES

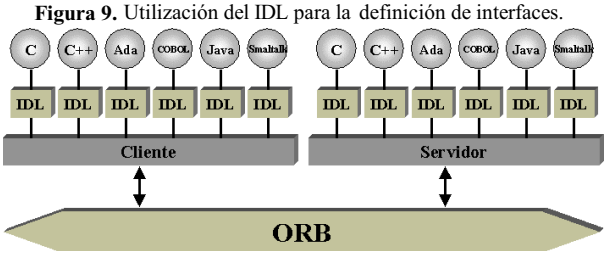
Antes de que un cliente haga una petición a un objeto, el cliente debe saber los tipos de operaciones que realiza el objeto. Es la interfaz del objeto la que especifica los métodos que el objeto está preparado para realizar, sus parámetros de entrada, su resultado y cualquier excepción que pueda generarse durante la ejecución.

CORBA utiliza el IDL (Interfaces Definition Languaje) para definir esas interfaces. Este lenguaje, es un lenguaje de especificación y no puede sustituir el papel de lenguajes de programación como pueden ser C++ y Java, ya que no es capaz de implementar las interfaces que especifica.

Gracias a la utilización de interfaces, se consigue la independencia del lenguaje de implementación, lo que permitirá al programador elegir el que más se adapte a sus conocimientos y necesidades.

El IDL aisla la implementación del objeto, que de esta forma se puede realizar en cualquier lenguaje de programación. En la figura 9 puede verse un esquema de su funcionamiento.

Figura 9. Utilización del IDL para la definición de interfaces.



Mediante el IDL, al igual que en cualquier otro lenguaje de programación, se especifican los tipos de datos que van a ser usados en la implementación. Además, el IDL permite el manejo de excepciones, que indican al programa su comportamiento cuando se producen situaciones anómalas a lo largo de la ejecución (división por cero, salidas de rango, etc.).

Par último señalar, que al ser el IDL un lenguaje de especificación, no se utiliza directamente para implementar los objetos, sino que es necesario traducirlo a los lenguajes de implementación. Esta función la realizan los ORB's que llevan incorporado su traductor de IDL a los lenguajes soportados en cada caso.

Hasta el momento la OMG ha especificado traducciones a los lenguajes de implementación C, C++, Ada, COBOL, Java y Smalltalk.

3.3 DESARROLLO DE APLICACIONES

La creación de aplicaciones en CORBA se trata de un proceso claramente definido tanto en las etapas que lo forman como en el orden de las mismas. Esto permite al desarrollador, una vez conocido el proceso, generar aplicaciones de forma rápida y sistemática. Los pasos a seguir son los que se muestran en la figura 10.

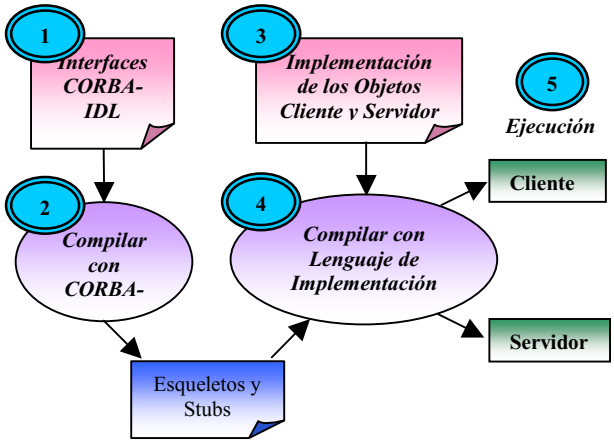


Figura 10. Pasos para el desarrollo de una aplicación CORBA.

- *Definir las interfaces de los objetos que van a participar en la aplicación mediante el IDL*. La interfaz de los objetos será su ventana al mundo. Permite que los demás objetos conozcan las operaciones que tienen disponibles y cómo deben ser invocadas. Mediante el IDL se definen: los métodos que el objeto exporta, sus parámetros, los atributos, las excepciones que genera, etc.
- *Compilar las interfaces de los objetos con el compilador CORBA-IDL*. Con la compilación del fichero de interfaces se obtienen los esqueletos del cliente y del servidor (en el lenguaje de implementación).
- *Crear la implementación de los objetos (servidor y cliente)*. Se crearán las clases que dan la

funcionalidad al objeto y las implementaciones del servidor y del cliente.

- *Compilar el código*. En este paso se procede a la creación de los ejecutables de la futura aplicación. Por un lado se compilará la implementación del servidor y la de los objetos con los esqueletos, para dar lugar a la parte servidora de la aplicación. Por otro lado, se compilará el cliente conjuntamente con los stubs para dar lugar a la parte cliente.
- *Ejecutar la aplicación*. La forma de ejecutar la aplicación distribuida dependerá del ORB que se esté utilizando. En algunos casos, previamente a la ejecución habrá que registrar el servidor en el almacén de implementaciones. En otros el registro será automático; y alternativamente, al registrar el servidor será el almacén de implementaciones el que arranque el servidor. Como norma general deberá arrancarse primero el servidor y posteriormente se podrá comenzar la ejecución del cliente.

4 SISTEMAS DE INFORMACIÓN INTELIGENTES

El rápido desarrollo de la Sociedad de la Información ha supuesto la aparición de nuevos conceptos como son las Empresas Virtuales, la Administración de la Cadena de Suministro, los Sistemas de Comercio Electrónico y los Sistemas Inteligentes de Organización y Acción.

Según la NIIIP (National Industrial Información Infraestructure Protocols Consortium), una Empresa Virtual es un consorcio temporal de empresas independientes que explotan conjuntamente las oportunidades de fabricación que surgen en el ámbito global, en un mundo rápidamente cambiante. En la figura 11 se muestra el nuevo marco competitivo global.

En el se representan las Empresas Virtuales Autónomas como objetos distribuidos (agentes inteligentes autónomos), que a su vez forman parte de objetos más extensos, como son las Cadenas de Suministro [2].

El desarrollo de aplicaciones según el estándar de CORBA facilita la construcción de agentes físicos siguiendo las recomendaciones de la FIPA (Foundation for Intelligent Physical Agents) para su aplicación industrial.

Empresas en sectores como el transporte, banca, telecomunicaciones y fabricantes; incluso gobiernos y universidades; han comenzado a beneficiarse de desarrollos con CORBA en sus sistemas de información [5]. Resultados comunes a todas ellas han sido el logro de importantes ventajas competitivas y reducciones en costes.

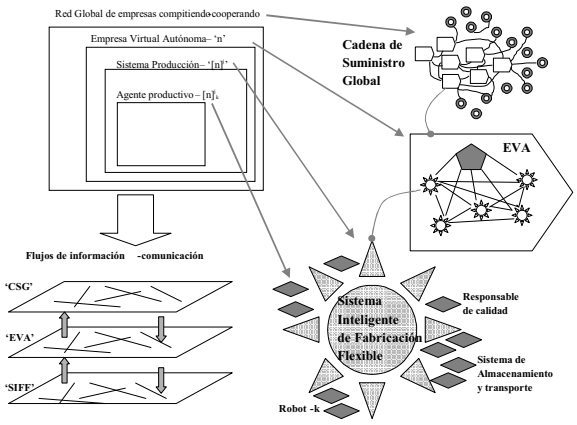


Figura 11. El marco competitivo de la Nueva Economía como sistema distribuido ([2]).

Agradecimientos

Este trabajo ha sido realizado gracias a la financiación recibida por el Ministerio de Educación y Cultura a través del proyecto PB47/0479.

Referencias

[1] López, A. (2000). *Análisis e Ingeniería de las Instituciones Económicas. Una Metodología Basada en Agentes*. Tesis Doctoral. Universidad del País Vasco.

[2] López, A. y Benito, J.J. de, (1999), FAMSA: Una Arquitectura de Sistemas de Fabricación Flexible en Empresas Virtuales Autónomas. En *Workshop Hipano Luso de Agentes Físicos* (Tarragona).

[3] Mowbray, T.J. y Ruh, W.A. (1997), *Inside CORBA, Distributed Object Standars and Aplications*. Massachusetts: Addison-Wesley.

[4] http://www.corba.org.

[5] Aguado, S. (2001). *Reingeniería de los Sistemas de Información con CORBA*. E.T.S. de Ingenieros Industriales. Valladolid.