# MODELO DE DESARROLLO DE APLICACIONES EN TRES CAPAS. CORBA Y HERRAMIENTAS DE DESAROLLO.

*Joaquín Peña, Francisco Leal, Rafael Corchuelo y Francisco Ferrer*
*Informe Técnico*
*Dpto. Lenguajes y Sistemas Informáticos de la Universidad de Sevilla*
*Avda. de la Reina Mercedes, s/n. Sevilla, 41.012*
*E-mail: jpena@lsi.us.es*

# INDICE

**RESUMEN** ..... **4**

**1. INTRODUCCIÓN** ..... **4**

1.1. INTRODUCCIÓN A CORBA ..... 5
1.2. HISTORIA DA LAS ARQUITECTURAS DE DESARROLLO DE SOFTWARE. ..... 7
*1.2.1.Aplicaciones monolíticas* ..... *7*
*1.2.2. Arquitecturas Cliente/Servidor* ..... *7*

**2. MODELO EN TRES CAPAS: 3-TIER ARCHITECTURE** ..... **8**

2.1. CAPA CLIENTE ..... 9
2.2. CAPA LÓGICA ..... 10
2.3. CAPA SERVIDOR ..... 10

**3. INTERCONEXION ENTRE CAPAS** ..... **10**

3.1. ENLACE AGENTE-SERVIDOR ..... 11
3.2. ENLACE CLIENTE-AGENTE ..... 11

**4. ALGUNAS HERRAMIENTAS QUE DAN SOPORTE AL MODELO EN TRES CAPAS** ..... **11**

4.1. DELPHI 4 ..... 12
4.2. MICROSOFT VISUAL STUDIO ..... 12
4.3. ORACLE 8 ..... 13
4.4. LOUIS ..... 15

**5. VENTAJAS DE LA ARQUITECTURA TRES-CAPAS** ..... **15**

**6. INCONVENIENTES DE LA ARQUITECTURA TRES-CAPAS** ..... **17**

**7. VENTAJAS DE LA UTILIZACIÓN DE CORBA EN SISTEMAS DE TRES CAPAS** ..... **18**

**8. CONCLUSIONES** ..... **20**

**9. AGRADECIMIENTOS** ..... **20**

**10. REFERENCIAS BIBLIOGRÁFICAS** ..... **21**

# TABLA DE ILUSTRACIONES

ILUSTRACIÓN 1. HETEROGENEIDAD DE CORBA ..... 5

ILUSTRACIÓN 2. ARQUITECTURA CORBA. ..... 6

ILUSTRACIÓN 3. ARQUITECTURA CLIENTE/SERVIDOR. ..... 7

ILUSTRACIÓN 4. MODELO DE DESARROLLO DE APLICACIONES EN TRES CAPAS ..... 9

ILUSTRACIÓN 5. APLICACIONES EN TRES CAPAS EN MSDS. ..... 13

ILUSTRACIÓN 6. APLICACIONES EN TRES CAPAS EN ORACLE. ..... 14

## RESUMEN

Hoy día, con la popularidad que está alcanzando Internet, es difícil imaginar un ordenador que no esté conectado a la RED (o a una intranet). El continuo avance tecnológico hace imprescindible el desarrollo de nuevas arquitecturas de desarrollo de software que se adapten a las nuevas necesidades. El modelo de desarrollo en tres capas surge de la necesidad de crear un software más competitivo y flexible y escalable. En este trabajo exponemos en qué consistente esta arquitectura, cuáles son sus ventajas e inconvenientes y hacemos un repaso a las principales herramientas comerciales que soportan esta forma de construir aplicaciones. También se muestra como la utilización del popular middleware CORBA reporta gran cantidad de ventajas a la hora de desarrollar aplicaciones con estas nuevas ideas.

## 1. INTRODUCCIÓN

Hace tiempo, el artista del *software* era una figura habitual en el mundo de la empresa. Se solía hacer referencia con este término a programadores expertos en resolver problemas complicados de una forma rápida que a menudo era él el único capaz de entender. La ingeniería del *software* siempre ha intentado luchar contra estos artistas proponiendo métodos adecuados para la construcción de buenas aplicaciones, entendiéndolas no sólo como aplicaciones correctas, robustas, eficientes y fáciles de mantener, sino también como aplicaciones que se desarrollan dentro de los márgenes establecidos en su presupuesto y permiten a la empresa que las realiza obtener beneficios.

La arquitectura de desarrollo de aplicaciones en tres capas surgió hace tiempo como una idea orientada a obtener mejores aplicaciones dotándolas de una mayor modularidad. Este factor, evidentemente, es uno de los que más repercute en la facilidad de reutilización, interoperatividad y, por lo tanto, en la facilidad para obtener aplicaciones adaptables a las necesidades y a los medios con los que cuentan clientes muy diferentes [Blaha *et al.*, 1997]. No obstante, el gran auge de esta arquitectura ha empezado a raíz de la popularización de Internet y la necesidad que tienen las empresas de satisfacer a sus clientes a través de un medio en el que el número de transacciones comerciales se multiplica con el paso de los días. En general, se observa que al igual que hace un tiempo todas las empresas querían añadir el calificativo orientado a objetos a sus productos, hoy casi todas desean poder desarrollar aplicaciones en tres capas.

En este trabajo introduciremos las características más novedosas de esta forma de construir aplicaciones, cuáles son las ventajas que podemos obtener y cuáles son los inconvenientes con los que podremos encontrar. También haremos un repaso de algunas herramientas comerciales que permiten aplicar estas ideas y apuntaremos otras formas de entender este concepto propuestas por otros autores. Además, presentaremos la forma en la que CORBA se relaciona con la arquitectura en tres capas y las ventajas que presenta su utilización.

En la introducción se hará una resumida introducción a CORBA, posteriormente se hará un breve recorrido por las arquitecturas de desarrollo de software de gestión a través la corta historia de la informática. Después, se introducirá la arquitectura en tres capas, sus ventajas e inconvenientes, una relación de las herramientas de desarrollo que podemos utilizar

## 1.1. Introducción a CORBA

CORBA es una norma, no un producto, que se encarga de especificar la interoperabilidad entre objetos en un entorno distribuido heterogéneo de manera transparente.

CORBA es una norma que fue impulsada por la OMG (Object Management Group) en 1990 [OMG, 1999] . Se propuso una arquitectura basada en componentes con capacidad de interoperabilidad (llevan integradas la conectividad de red) con independencia de la localización. Finalmente en 1992 la OMG definió el primer ORB (Object Request Broker).

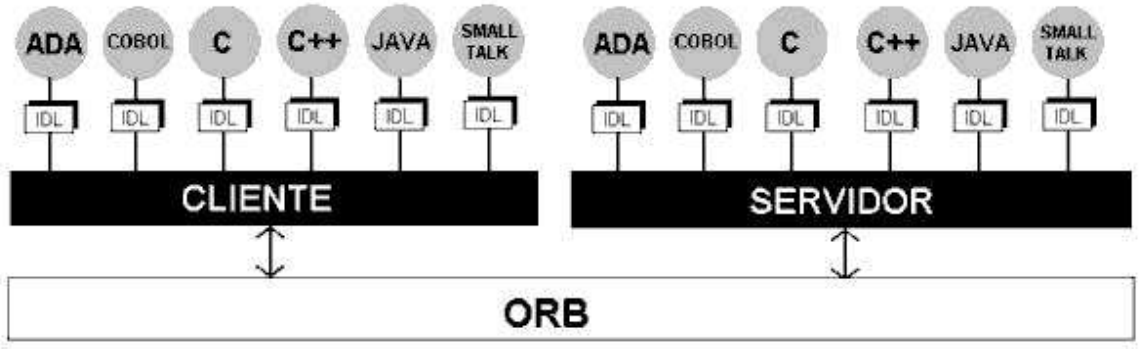


**Ilustración 1. Heterogeneidad de CORBA.**

CORBA en definitiva, es un middleware que permite conectar clientes y servidores heterogéneos, tanto en el lenguaje de programación como en la plataforma de ejecución. Esto se consigue de forma transparente para los clientes y servidores. Los clientes simplemente invocan un método de un objeto servidor y no tienen que preocuparse de la localización de dicho objeto. Además, la petición se realiza en el lenguaje de programación en que está codificado el cliente aunque el servidor esté realizado en otro distinto. El encargado de la traducción y transporte de la llamada es el ORB (Object Request Broker), una pieza fundamental de la arquitectura de CORBA.

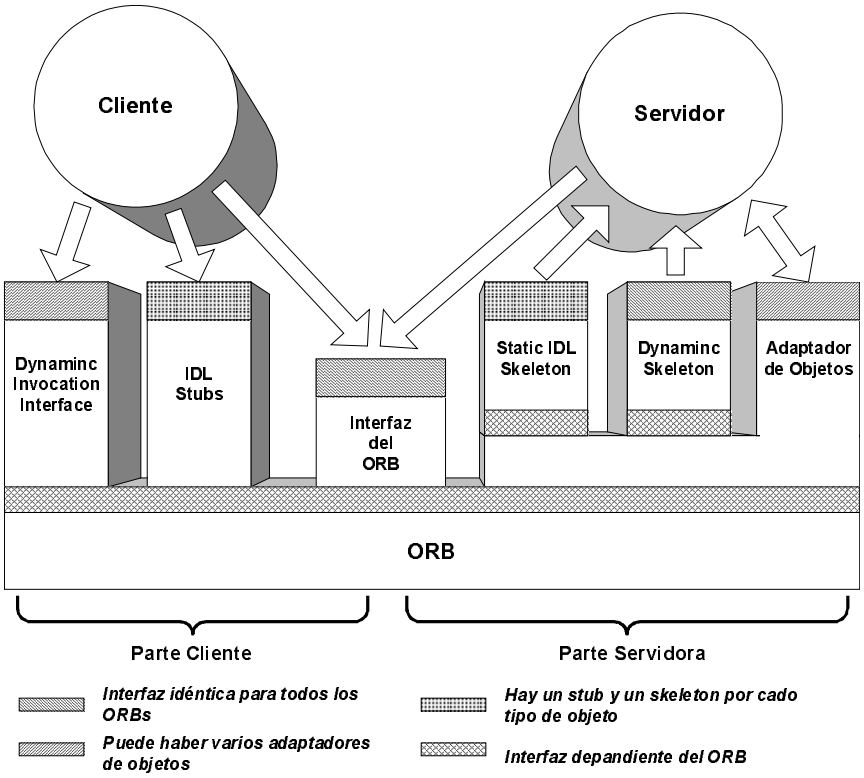


**Ilustración 2. Arquitectura CORBA.**

El ORB, ver ilustración 2, es el "bus de objetos", lo podríamos comparar con el Software de TCP/IP que permite a los mismos hacer llamadas a otros de forma trasparente, independientemente de su localización, del sistema operativo donde se ejecutan o del lenguaje en el que están codificados.

Esta magia se consigue gracias al IDL (Interface Definition Language) que es una especie de Esperanto de los lenguajes de programación. Mediante IDL se especifica qué servicios (el interface que posee) ofrece cada objeto servidor. Usando un compilador de IDL, se genera el código necesario en un lenguaje de programación determinado para el lado cliente (los stubs) y el lado servidor (skeletors) que implementa dicha funcionalidad.

Es decir, en la parte cliente el código generado se usa en la implementación del cliente como cualquier librería convencional (por ejemplo, una librería para el manejo de Sockets) que hace ver al cliente el objeto(s) servidor y sus servicios.

En la parte servidora se genera código que recibe las peticiones locales o vía red de los clientes y se encarga de llamar a las funciones correctas del servidor cuyas cabeceras/esqueletos también se han generado, lógicamente debiendo ser implementado el código de los métodos por el programador. Es decir, nos encontramos básicamente con un esquema similar al conocido de las clásicas Remote Procedure Calls.

## 1.2. Historia da las arquitecturas de desarrollo de software.

### 1.2.1.Aplicaciones monolíticas

Antes de aparición de arquitecturas de alto nivel como la que estamos tratando, el desarrollo de aplicaciones era bien distinto. El programador no se ceñía a ningún modelo determinado, sino que tomaba un lenguaje de programación, COBOL en la mayoría de los casos, y comenzaba a programar y depurar. El resultado era una aplicación monolítica, en la que se mezclaba la interfaz de usuario con los datos y la lógica de negocio, de modo que al estar todo entrelazado, cualquier cambio suponía un gran esfuerzo. De todos modos, pese a los inconvenientes que esta técnica conlleva, hoy día hay gran cantidad de empresas que siguen usando sistemas aplicaciones monolíticas, aunque muy depuradas y adaptadas a las necesidades actuales, ya que los costos de rehacer la aplicación de nuevo con las técnicas actuales son demasiado elevados.

### 1.2.2. Arquitecturas Cliente/Servidor

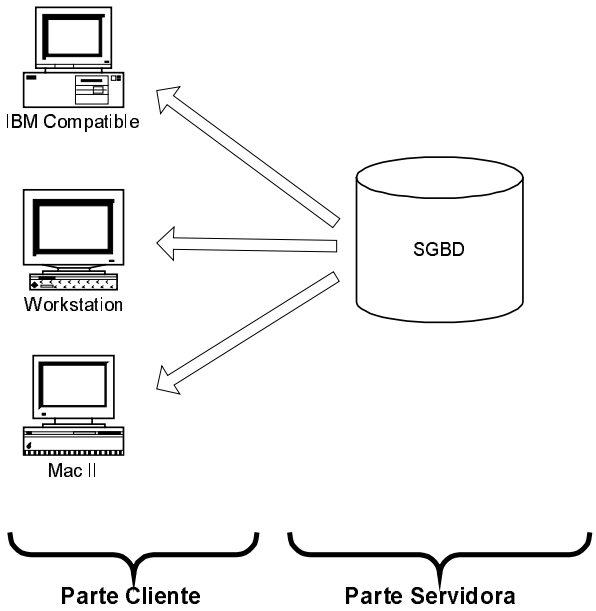


**Ilustración 3. Arquitectura Cliente/Servidor.**

Posteriormente apareció la llamada arquitectura Cliente/Servidor. La arquitectura Cliente/Servidor presenta muchas ventajas con respecto a las aplicaciones monolíticas. Como se observa en la ilustración 3, en un sistema Cliente/Servidor se distinguen dos tipos de nodos: los nodos Clientes, a la izquierda en la figura, y el o los nodos Servidores, a la derecha en la figura. Los Clientes (se utiliza él termino en plural debido a que generalmente suele haber más de un Cliente) son los que generalmente interactúan

con el usuario y proporcionan la interfaz gráfica para recoger las peticiones del usuario y comunicárselas al Servidor; por otro lado, el Servidor es el que se encarga de almacenar la información (DBMS sistema de gestión de la base de datos) y proporcionársela al Cliente cuando éste así lo requiera.

En la mayoría de los casos la gestión, traducción, etc. de las peticiones, y por consiguiente la mayor parte del peso de computación y administración de la aplicación, recaen sobre el Cliente.

La arquitectura Cliente/Servidor es la más usada en la actualidad, ya que presenta muchas mejoras con respecto a las aplicaciones monolíticas. Por el contrario la organización Cliente/Servidor en dos capas nos plantea también serias desventajas:

Para empezar, el Cliente y el Servidor estarán escritos para una determinada plataforma y un determinado gestor de bases de datos, por lo que el código no será reutilizable en otros casos, y si se quiere realizar algún cambio se tendrá que reescribir la aplicación completa.

La encapsulación de los datos (ocultación de la información) es pobre a causa de que el Cliente debe conocer todas las características del Servidor y la forma en que se almacena y trata la información.

Otro de los inconvenientes que se presenta es que se ve limitada tanto la escalabilidad horizontal del sistema como la escalabilidad vertical. La escalabilidad horizontal consiste en aumentar el número de Clientes, y si éste aumenta demasiado puede incluso producirse un colapso del Servidor (el número de clientes es limitado). La escalabilidad vertical, al consistir en migrar el Servidor a un sistema más potente, por ejemplo con varios Servidores o un DBMS más potente, puede obligar a reescribir la aplicación desde cero.

Además se debe observar que en la mayoría de los casos es en el cliente donde se encuentra la mayor parte de la lógica de administración del sistema, por lo que la cantidad de código será grande y si se accede a éste a través de Internet la descarga de la aplicación cliente puede llevar bastante tiempo.

Por último, la administración y la seguridad de estos sistemas es compleja al estar la lógica de la aplicación distribuida entre el cliente y el servidor y no poder tratarla de una manera centralizada.

## 2. MODELO EN TRES CAPAS: 3-TIER ARCHITECTURE

El modelo en tres capas surge como una mejora sobre la arquitectura clásica en dos capas. Esta basado en la inclusión de una nueva capa o nivel de abstracción en la que se aísla la lógica de administración del sistema del resto de bloques.

El objetivo de esta división es hacer cada capa completamente independiente de las demás y hacer que cada una de ellas se comporte de manera totalmente transparente,

es decir, que las características de una capa concreta no dependan en ningún aspecto del resto de las capas. Así, cada capa se ve como un elemento independiente del resto.

Una de las consecuencias de esta mejora es que la parte cliente queda reducida a una simple Interfaz gráfica. Por otro lado, la parte servidora quedará reducida a la base de datos y el DBMS y por último, se centralizará en la capa que se inserta la lógica de negocio de la aplicación.

Gracias a la versatilidad que nos da este modelo podremos utilizar cada capa como piezas de un mecano tomando el número de ellas que necesitemos adaptando el resultado final a nuestras necesidades. Además, se podrá utilizar cada pieza todas las veces que sea necesario.

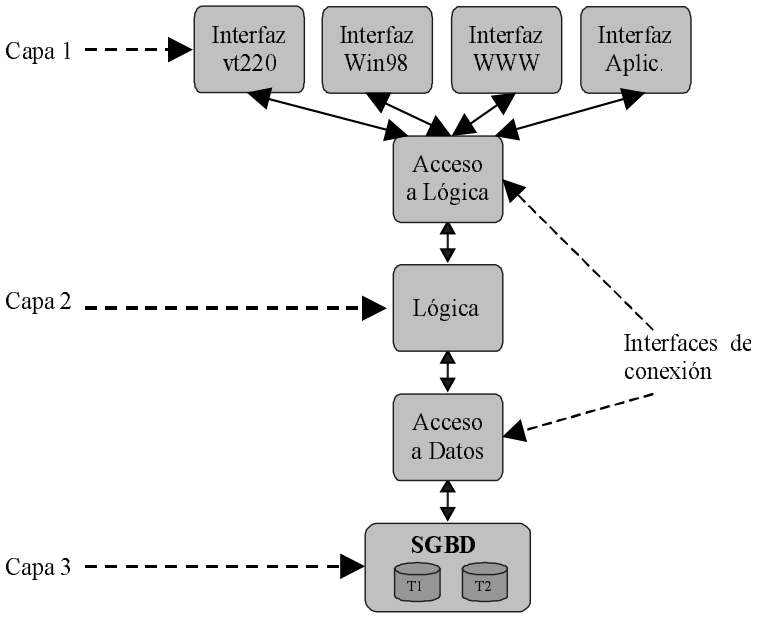


**Ilustración 4. Modelo de desarrollo de aplicaciones en tres capas.**

Como se puede observar en la ilustración 4 la arquitectura consta de varias capas que explicaremos a continuación:

## 2.1. Capa Cliente

La primera capa que se observa en la figura es el cliente. En ella estará implementado el cliente en forma de interfaz gráfica que interactuará con el usuario pidiendo datos de entrada y mostrando los datos que produce la aplicación. La interfaz de usuario puede consistir desde una aplicación independiente hasta un terminal o incluso una pagina Web.

La misión de esta capa es únicamente contener la Interfaz gráfica para interactuar con el usuario.

Además, si se tienen distintos tipos de clientes (encargados de sección, cajeros...) con necesidades distintas y plataformas distintas, la cantidad de código a desarrollar será pequeña y sencilla ya que sólo habrá que escribir la interfaz de usuario. Si se utiliza un editor de interfaces gráficas la tarea se hace casi trivial.

### 2.2. Capa Lógica

La nueva capa que introduce el modelo, es la capa intermedia y como ya hemos apuntado es la encargada de manejar la lógica de la aplicación, de forma que la implementación de esta capa deberá ser lo más abstracta posible con el objetivo de que no afloren detalles de implementación.

Esta capa se verá como una especie de interfaz que nos proporcionará una serie de primitivas y datos a los que podremos acceder. De esta forma, se dispondrá de una serie de servicios que podrán ser usados por la capa cliente. El cliente deberá poder ser programado sin necesidad de conocer la implementación interna que se haga de la lógica de negocio con el objetivo de que las modificaciones realizadas a la capa lógica mantengan la interfaz (la apariencia externa) intacta y no desemboquen en la modificación de ninguna de las otras dos capa.

La principal función del agente es la de traducir y adaptar las peticiones del cliente (llamadas a las primitivas que proporciona la capa) a la semántica interna del servidor.

Además, gestiona el acceso de los clientes al servidor, y al estar esta tarea centralizada en esta capa, aumenta la seguridad y la eficiencia, ya que se puede hacer una mejor gestión de la carga del servidor. Por ejemplo, se pueden introducir varios servidores y balancear la carga entre ellos, con lo que conseguimos que el número de clientes puede aumentar considerablemente.

### 2.3. Capa Servidor

Esta capa consiste en un gestor de bases de datos (DBMS). Hay que notar que, al existir muchos gestores de bases de datos distintos, cada uno de ellos posee unas rutinas de acceso diferentes y esto haría que nuestra capa lógica no fuera totalmente abstracta y dependiera fuertemente del gestor que utilicemos. Como veremos posteriormente, este problema se puede salvar gracias a SQL con el que, al estar estandarizado y existir puentes que traducen sentencias SQL a las propias del gestor, podremos trabajar de manera independiente.

## 3. INTERCONEXION ENTRE CAPAS

Otro de los elementos del modelo muy a tener en cuenta es la manera en la que se comunican las capas entre sí. La interfaz de usuario se comunicará con el agente, y éste a su vez lo hará con el gestor de base de datos. Así pues, tenemos dos vías de co-

municación, que tendrán objetivos y necesidades distintas, por lo que se utilizarán elementos distintos para cada una de ellas.

### 3.1. Enlace Agente-Servidor

Los fabricantes de bases de datos suelen proporcionar rutinas de acceso al gestor de la base de datos, pero aunque el acceso es más rápido no es aconsejable utilizarlas si se quiere mantener la flexibilidad del sistema. Hay otras formas de comunicar al agente con la bases de datos de manera que no se tengan que utilizar las rutinas determinadas de cada gestor. Esto se puede lograr utilizando ODBC o JDBC, consistentes en realizar todas las consultas contra la bases de datos en SQL. Será ODBC o JDBC, según el caso, el encargado de traducir las peticiones a la semántica propia del gestor que se esté utilizando. Esto permite cambiar el sistema gestor de bases de datos con relativa facilidad, ya que no habrá que cambiar prácticamente nada en el código (sólo la línea en que se especifica el gestor utilizado en el caso de JDBC) del Agente.

### 3.2. Enlace Cliente-Agente

Para comunicar el cliente con el agente existen varias posibilidades, algunas de ellas son Java RMI, DCE, DCOM [MS1, 1999] y CORBA [OMG, 1999]. Si se quiere conseguir un producto flexible, portable y escalable, son estas dos ultimas la más adecuadas ya que, por ejemplo, la primera alternativa nos obliga a utilizar Java en las dos partes. Teniendo en cuenta que hasta la actualidad DCOM es un producto que sólo está disponible para plataformas Windows, aunque Microsoft ha anunciado que en los próximos años empresas como Hewlett Packard, Siemens-Nixdorf o Silicon Graphics, entre otras, proporcionarán implementaciones de DCOM para sus versiones de UNIX [MS1, 1999] , en cambio CORBA está disponible para un gran número de plataformas y lenguajes de programación. CORBA nos permite utilizar gran multitud de lenguajes de programación (existen implementaciones de CORBA para C++, JAVA, Smalltalk, Ada 95, COBOL) con lo que el código del cliente y el del agente pueden estar escritos en lenguajes diferentes y sobre plataformas diferentes. Con el uso de CORBA, incluso se podrá cambiar el cliente o el agente sin necesidad de reescribir la otra capa.

## 4. ALGUNAS HERRAMIENTAS QUE DAN SOPORTE AL MODELO EN TRES CAPAS

A la hora de analizar las herramientas que dan soporte a este modelo debemos establecer una clara diferencia entre aquellas que permiten realizar nuevas aplicaciones a partir de cero y aquellas otras que, además, nos permiten reutilizar viejas aplicaciones escritas en COBOL o RPG desarrolladas por algún artista del *software* tiempo atrás. Esta cuestión es clave, ya que la mayoría de las empresas interesadas en las ventajas que este modelo supone tienen ya sus aplicaciones desarrolladas utilizando el modelo monolítico o cliente/servidor y, evidentemente, no es razonable pensar que van a poder acometer la inversión que supondría volver a implementarlas haciendo uso de nuevas herramientas.

En esta sección analizaremos algunas alternativas con bastante éxito en el mercado y comentaremos cuáles son sus características más innovadoras.

### 4.1. Delphi 4

Delphi 4 [Inprise, 1999] es una herramienta de desarrollo rápido de aplicaciones que cuenta con una gran aceptación en el mercado del *software* de gestión. Proporciona un entorno de trabajo para Windows que ofrece facilidades de conexión con la mayor parte de los gestores de bases de datos que se encuentran en el mercado y además soporta de manera natural la utilización de CORBA, ya que el ORB Visibroker pertenece al igual que Delphi a la empresa Inprise. Por otra parte, la biblioteca de componentes que proporciona permite obtener con facilidad interfaces de usuario atractivos con un esfuerzo mínimo.

En su versión anterior proporcionaba ya la posibilidad de realizar aplicaciones en tres capas utilizando DCOM y ODBC como interfaces de conexión. La innovación más importante de la versión cuatro es la posibilidad de utilizar CORBA, por lo que ha roto la barrera de los sistemas Windows y facilita el desarrollo de aplicaciones distribuidas en entornos heterogéneos.

El problema fundamental es que no está preparada para adaptar aplicaciones ya desarrolladas a la arquitectura en tres capas. Por desgracia, en los programas desarrollados con aplicaciones anteriores la lógica está tan entrelazada con la interfaz de usuario, que separar las dos capas puede exigir un esfuerzo en muchas ocasiones superior al de realizar la aplicación desde cero.

### 4.2. Microsoft Visual Studio

Microsoft propone la herramienta Visual Studio [MS2, 1999] para el desarrollo de aplicaciones en tres capas a partir de cero. En esta herramienta, la construcción de una aplicación se entiende como un proceso de desarrollo de componentes abstractos capaces de realizar tareas específicas y de comunicarse con otros componentes para completar su funcionalidad.

La ilustración 5 muestra de forma esquemática la forma en que se construye una aplicación con esta herramienta. La base es un conjunto de componentes que se pueden implementar en cualquiera de los lenguajes que ofrece este entorno de trabajo: Visual Basic, Visual C++, Visual J++ e incluso COBOL y Pascal. En cualquiera de los casos, el enlace entre los diferentes componentes se produce de forma dinámica, prescindiendo por completo de los detalles de implementación, lo que permite que un componente pueda ser reemplazado sin necesidad de volver a compilar y distribuir toda la aplicación. Como interfaz de comunicación entre las distintas capas Microsoft ofrece la posibilidad de usar COM, DCOM, ODBC y JDBC, por lo que el abanico de posibilidades es bastante amplio.

Las dificultades para mantener la integridad de los datos y la fiabilidad del sistema se pueden superar gracias al Microsoft Transaction Server (MTS), que garantiza la

sincronización en el acceso simultáneo a las bases de datos e incorpora el standard SSL para proteger la información que fluye a través de la red de miradas no autorizadas. MTS controla la creación, el mantenimiento y la destrucción de componentes optimando el consumo de recursos. Por otra parte, esta herramienta también facilita una distribución más flexible de los datos ya que los módulos de interfaz se comunican con ella sin necesidad de acceder de forma directa a los componentes de la aplicación o al sistema de gestión de bases de datos.

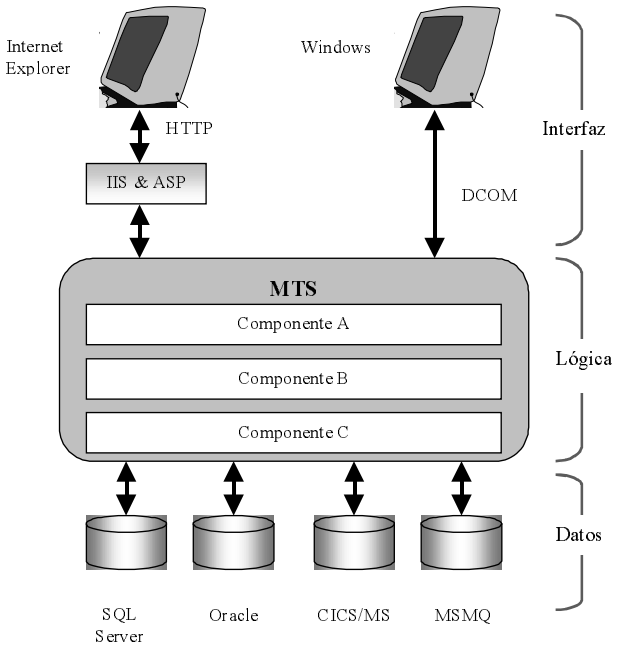


**Ilustración 5. Aplicaciones en tres capas en MSDS.**

Las posibilidades de acceso a las aplicaciones a través de Internet también son bastante buenas mediante el uso del Internet Information Server, que permite a los usuarios utilizar su navegador *Web* para acceder a los componentes de la aplicación mediante el uso de ASP.

Como hemos podido observar, es una herramienta muy preparada para facilitar al máximo la interoperatividad entre componentes en entornos distribuidos, pero está pensada para desarrollar aplicaciones a partir de cero y no ofrece facilidades para reaprovechar aplicaciones antiguas desarrolladas utilizando el modelo monolítico, por ejemplo.

### 4.3. Oracle 8

En su versión 8, Oracle ha dado el gran salto hacia la interconectividad total [Oracle, 1999]. Es la última generación de uno de los sistemas de gestión de bases de datos líder en el mercado y es el primero que ha sido desarrollado íntegramente mirando a aprovechar las grandes posibilidades que Internet proporciona a las empresas. Por esta razón, no es de extrañar que la posibilidad de desarrollar aplicaciones en tres capas sea uno de sus puntos fuertes.

Las herramientas de desarrollo que ofrece permiten crear la interfaz de la aplicación en Java, de forma que este se pueda ejecutar con facilidad dentro de una página *Web*. Gracias a un conjunto de rutinas específicas, esta interfaz es capaz de comunicarse con la capa de lógica que reside en lo que Oracle denomina servidor de aplicaciones, que no es más que una máquina virtual, en el sentido descrito anteriormente, que se encarga de ejecutar los servicios que le piden sus clientes de la forma más efectiva que resulte posible. Los componentes de las aplicaciones pueden desarrollarse en una amplia gama de lenguajes que incluye Java, Perl, C++ o PL/SQL.

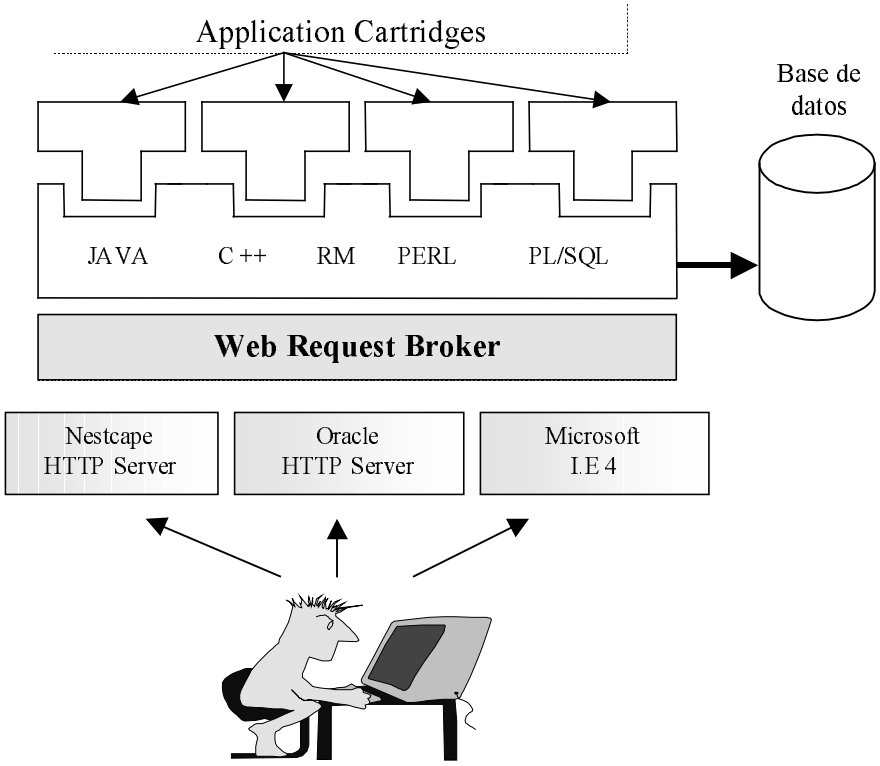


**Ilustración 6. Aplicaciones en tres capas en Oracle.**

La ilustración 6 recoge de forma esquemática el aspecto de una aplicación desarrollada en Oracle. El elemento fundamental es el Web Request Broker (WRB), un módulo que juega el papel de interfaz entre los componentes de las aplicaciones y la interfaz de usuario que se utiliza para acceder a ellas. En su versión actual, el WRB está disponible para Windows NT y UNIX utilizando Internet Explorer o Netscape. El método de comunicación entre los componentes es mediante el uso de una interfaz específica denominada Inter-Cartridge Exchange (ICX) que proporciona un conjunto de APIs para la comunicación y el acceso compartido a los datos. Además, se ha prestado especial atención a la interoperatividad, razón por la que todos los cartuchos y servicios estándares se han implementado como objetos distribuidos CORBA. Esto, además, permite distribuir las aplicaciones en entornos heterogéneos con una enorme facilidad. En cuanto a la protección de datos, Oracle 8 incorpora la posibilidad de usar SSL.

En resumen, el servidor de aplicaciones proporciona una plataforma que combina la potencia y fiabilidad de los sistemas tradicionales con la flexibilidad y facilidad de uso de Internet. Por otra parte, Oracle 8 también es capaz de gestionar el orden en que

se deben atender las diferentes peticiones garantizando la seguridad y optimando el uso de los recursos disponibles en cada momento. Buscando aumentar al máximo el rendimiento de esta nueva versión, también se han introducido notables mejoras en la capa de almacenamiento de datos. Algunas de ellas, como el uso de punteros, tablas anidadas o ciertas características de orientación a objetos permiten acelerar enormemente las consultas de la base de datos, por lo que, en conjunto, es una de las herramientas que está contando con una mejor aceptación.

### 4.4. LOUIS

Finalmente, en este pequeño repaso que hemos realizado, no debemos dejar de lado a LOUIS [Softis, 1999]. A diferencia de Delphi, Visual Studio u Oracle, LOUIS no es un entorno de desarrollo, sino una biblioteca de tipos abstractos de datos disponible para un gran número de plataformas y lenguajes de programación diferentes.

El tipo de datos fundamental es LOUIS–VALUE, que sirve para modelar los datos de una forma completamente independiente al lenguaje en que está escrita la aplicación que hace uso de ellos, la plataforma sobre la que se ejecuta o el tipo de red a través del que se van a transmitir. Mediante este tipo abstracto de datos podemos encapsular la información que aparece en una pantalla, transmitirla a la capa de lógica, procesarla y devolver resultados a la capa de interfaz de usuario mediante otro tipo de datos llamado LOUIS–COMMUNICATOR.

LOUIS es por lo tanto una interfaz de interconexión entre la capa de interfaz y la lógica que tiene su principal punto fuerte en el hecho de que está disponible para un amplísimo rango de plataformas y lenguajes de programación, lo que facilita el desarrollo de la lógica en COBOL y el acceso a la misma desde un programa escrito en Java, por ejemplo. Otro de sus puntos fuertes es que es una de las pocas herramientas capaces de **reaprovechar** con muy poco esfuerzo **aplicaciones ya desarrolladas** sin pensar en absoluto en las ventajas que ofrece la arquitectura en tres capas. En concreto, viene acompañada de una herramienta llamada CLARK que permite analizar un programa escrito en COBOL y separar la capa de lógica de la de interfaz de usuario de una forma completamente automática. Las interfaces de usuario que se obtienen de esa forma no son comparables a los que se pueden desarrollar con Visual Basic o Delphi, pero al menos permite establecer una clara separación entre las dos capas, por lo que podremos reaprovechar la lógica de la aplicación sin esfuerzo alguno y con completa seguridad.

## 5. VENTAJAS DE LA ARQUITECTURA TRES-CAPAS

Hasta este punto hemos introducido algunas de las ventajas que nos proporciona la arquitectura en tres capas con CORBA, ahora veremos una pequeña síntesis de todas estas:

La **reutilización** es una de las principales ventajas que ofrece este modelo. Al ser todos las capas independientes de las demás, se podrá retocar cualquiera de ellas para mejorarla, e incluso reescribirla de nuevo si es necesario, sin necesidad de cambiar las otras.

Otra de las ventajas, es que hace que aumente la **escalabilidad**. El modelo de tres capas permite escalar nuestros sistemas con mayor facilidad que otros. Normalmente, una empresa tiene clientes de tipos muy diferentes, con equipos de distinta potencia, que hará necesario que se puedan adaptar las interfaces de usuario a todas las máquinas, tanto a las más antiguas como a las más recientes. La arquitectura en tres capas proporciona la ventaja de poder desarrollar distintos clientes que se adapten a las distintas necesidades hardware, aprovechando de esta forma todos los recursos disponibles y evitando a la empresa la costosa compra de equipos nuevos.

La arquitectura en tres capas permite **distribuir** la aplicación ejecutando cada capa en la máquina que ofrezca las mejores características. Al ser cada capa independiente de las demás, se podrá alojar cada capa donde queramos preocupándonos sólo de que se comunique con el resto de manera adecuada.

Una ventaja importante es la **heterogeneidad**. Al poder adaptar cada capa al hardware, sistema operativo, etc. de la máquina en que reside cada capa, esta arquitectura puede sobrevivir en un entorno heterogéneo.

Adaptar las aplicaciones a **Internet** es sin duda un de los objetivos más perseguidos actualmente por las empresas, que quieren ofrecer sus servicios a través de WWW. Las dos últimas características, distribución y heterogeneidad, anteriormente citadas son quizás las más relacionadas con Internet. La WWW ofrece un entorno completamente heterogéneo al que tienen acceso desde potentes estaciones de trabajo hasta PC personales, pasando por Macintosh e incluso teléfonos móviles (ya podemos programar interfaces gráficas en Java para teléfonos móviles, que funcionen con micros que ejecuten código Java directamente). Además se pueden establecer conexiones desde cualquier lugar del mundo, así que debemos disponer de aplicaciones capaces de funcionar eficiente mente a través de la red.

Con la separación entre interfaz gráfica y lógica de negocio que hace la arquitectura en tres capas se obtiene la principal baza para la inmersión de una aplicación en Internet. Se Puede, por ejemplo, utilizar para la interfaz gráfica un applet Java incrustado en una página WEB. Al ser el cliente una pagina WEB, los usuarios de esta página podrán utilizar cualquier Sistema Operativo y cualquier plataforma (heterogeneidad), lo único que necesitarán será un navegador. Así, el usuario siempre encuentra la misma Interfaz gráfica aunque cambie de plataforma, navegador, y los cambios en el cliente sólo habrá que hacerlos en el servidor http, y no habrá que ir cliente por cliente actualizando el sistema.

La arquitectura en tres capas reduce **costes de mantenimiento**, ya que al tratarse de una arquitectura que aísla cada nivel lógico en una capa, el mantenimiento se hace mucho más sencillo. El **código** esta mucho más **estructurado** que en las aplicaciones monolíticas y cliente/servidor, de forma que su **comprensión** es mucho más fácil. Como cada concepto esta totalmente aislado de los demás se puede localizar con facilidad el aspecto que se quiere modificar y realizar el cambio sin afectar al resto de capas.

Otro problema que se mejora en ciertos aspectos es la **eficiencia**. Normalmente en un sistema de bases de datos hay consultas que se realizan con relativa asiduidad y que producen todas el mismo resultado. Si se introduce en la capa del agente una cache

que responda a estas consultas sin tener que acceder al DBMS se descargará a este y las respuestas se realizaran de manera más eficiente (sólo se actualizará los datos de la cache cuando sea necesario).

## 6. INCONVENIENTES DE LA ARQUITECTURA TRES-CAPAS

Las ventajas, como acabamos de comprobar, son muy atractivas, pero lo cierto es que a la hora de aplicar en la práctica el modelo en tres capas nos podemos encontrar con problemas técnicos que hoy en día aún no han sido completamente resueltos por todas las herramientas de desarrollo. En esta sección comentaremos algunos de ellos:

Uno de los problemas más importantes son los aspectos de **seguridad**. Cuando las aplicaciones se desarrollaban haciendo uso del modelo monolítico, los únicos problemas de seguridad se reducían a que alguien no autorizado tuviese acceso a nuestro ordenador. Ahora, desde el momento en que se recuperan los datos del almacenamiento hasta que la información elaborada llega al usuario debe pasar por la red y por las interfaces de conexión entre las tres capas. Esto hace que no sea suficiente con una herramienta capaz de interconectar varias máquinas de una forma más o menos simple, además, debe ofrecernos ciertas garantías de que esa información no va a ser observada o manipulada de forma no autorizada.

CORBA ofrece un servicio de **encriptación** de información al poder utilizar el protocolo IIOP sobre SSL, de forma que podemos solventar el problema si utilizamos Delphi, además, Visiborker de Delphi proporciona herramientas que permiten la utilización conjunta de CORBA con Firewalls. LOUIS no ofrece la posibilidad de encriptar información y delega estas tareas en el sistema operativo. De este modo, Delphi, Microsoft Developer Studio y Oracle son las únicas herramientas que hemos analizado que proporcionan servicios específicos para proteger la información que fluye a través de la red.

Por otra parte, al hacer un **uso más intensivo de la red** de comunicaciones, el ancho de banda que requieren nuestras aplicaciones se multiplica, lo que significa que si gradualmente desarrollamos más y más aplicaciones haciendo uso de este modelo, no es de extrañar que en un plazo de tiempo no demasiado largo la red de interconexión se convierta en un cuello de botella importante.

Generalmente, las herramientas que permiten crear aplicaciones en tres capas permiten al diseñador determinar cuál es el tamaño de los paquetes de datos que se van a transmitir por la red, pero este dato suele fijarse en tiempo de diseño, algo que no es del todo adecuado si tenemos en cuenta que el volumen de datos que maneja una aplicación puede variar con el tiempo dependiendo de factores a los que el diseñador puede ser completamente ajeno.

De las herramientas que hemos estudiado, tan sólo Oracle promete características adecuadas para optimar el uso de la red de comunicaciones, pero, por desgracia, nos ha sido imposible evaluar estos aspectos en la práctica.

Tampoco debemos olvidar que en una aplicación monolítica o cliente/servidor, la lógica de los programas consume potencia local dentro de cada una de las máquinas en las que se ejecuta. Ahora, en teoría sería una única lógica la responsable de responder a todas las peticiones que realizasen las distintas interfaces de usuario, algo que evidentemente no es aconsejable si esa aplicación debe dar servicio a un número de usuarios elevado. Lo más adecuado para resolver este problema sería poder contar con varias máquinas encargadas de la lógica de tal forma que el **equilibrio de la carga** de trabajo se realizase de una forma automática entre ellas.

Visual Studio o Delphi 4 permite distribuir los componentes que forman la lógica con cierta facilidad, pero ninguna de las dos herramientas proporciona medios para distribuir la carga de trabajo de manera directa. Sin embargo, si usamos el ORB que proporciona Delphi, existen herramientas que permiten balancear la carga de manera automática. Oracle es quizá el mejor preparado en este sentido, ya que al estar completamente basado en CORBA permite distribuir con gran facilidad las aplicaciones entre varias máquinas. Por otra parte, tanto Visual Studio y Oracle están especialmente preparados para atender las peticiones de forma óptima de tal manera que se pueda sacar el máximo partido a los recursos de la máquina en que se ha instalado el servidor de aplicaciones.

En general, no es fácil **convertir** una aplicación ya desarrollada que utiliza el modelo monolítico o cliente/servidor **en un aplicación en tres capas**. Se ha desarrollado bastante esfuerzo para obtener metodología que nos permitan dar el paso de una forma segura y poco costosa [Noffsinger *et al.*, 1998; Shoffner, 1998], pero no es habitual encontrar soporte en las herramientas de desarrollo para reconvertir aplicaciones.

De entre las que hemos analizado en este trabajo, tan sólo LOUIS incorpora una herramienta para separar la lógica del interfaz en aplicaciones escritas en COBOL. Esto nos permite reutilizar con facilidad viejas aplicaciones con un mínimo esfuerzo y con todas las garantías de que la aplicación resultado funcionará exactamente igual que la original.

## 7. VENTAJAS DE LA UTILIZACIÓN DE CORBA EN SISTEMAS DE TRES CAPAS

Mediante la utilización de CORBA en un sistema de tres capas se obtienen un buen número de ventajas, entre ellas las siguientes:

CORBA permite **conectar clientes y servidores heterogéneos**, tanto en lenguajes de programación como en plataformas de ejecución. Esto proporciona una gran flexibilidad a la hora de adaptar una aplicación a un entorno en el que existan diferentes necesidades hardware y software.

La **comunicación** se realiza de forma **transparente**. El cliente simplemente obtiene una referencia a un objeto y puede invocar cualquiera de sus métodos sin tener que preocuparse sobre el transporte, la localización de los servidores, la activación de los objetos, el orden de los bits o el sistema operativo que utilice. Además, las llamadas a

estos métodos se realizan en el lenguaje de programación del cliente, independientemente del lenguaje en el que esté implementado el servidor.

CORBA permite la posibilidad de **conectar varios ORB** a través del protocolo IIOP (Internet Inter-ORB Protocol) [J. Peña, 2000]. Un ORB puede manejar llamadas entre objetos dentro de un proceso simple, múltiples procesos corriendo en la misma máquina o múltiples procesos corriendo sobre varias redes y sistemas operativos. Esto es completamente transparente a nuestros objetos.

El ORB incluye información contextual en sus mensajes para manejar la seguridad y las transacciones a través de los límites de la máquina y del ORB.

Otra de las ventajas que ofrece CORBA (una de las más atractivas) consiste en que es una **solución evolutiva**, ya que permite la coexistencia con sistemas ya existentes en un determinado entorno. La separación que proporciona CORBA entre la definición y la implementación de un objeto es la opción ideal para encapsular aplicaciones ya existentes. Usando CORBA IDL (Interface Definición Language) se puede crear una interfaz de forma que el código ya existente aparezca como un objeto para el ORB, incluso si está implementado con lenguajes no orientados a objetos como COBOL. Gracias a esto se puede trasladar una aplicación monolítica a una arquitectura de tres capas aprovechando el código ya escrito. Para aclarar este concepto, se verá el siguiente ejemplo:

*Un banco tiene una base de datos cuya gestión se realiza mediante una aplicación escrita en COBOL. Debido a la apertura de nuevas sucursales se hace necesario comunicar la aplicación con más terminales, pero se produce una gran sobrecarga del servidor. El banco decide cambiar la aplicación por otra que se adapte a las nuevas necesidades, pero manteniendo la misma base de datos y a ser posible la misma interfaz de usuario. Se pide presupuesto a una empresa software para desarrollar la nueva aplicación, pero se desecha el proyecto debido a su elevado precio. Como segunda opción, se propone al banco una solución alternativa, que consistente en modificar la aplicación antigua trasladándola a una arquitectura en tres capas. El objetivo es tomar la antigua aplicación, separar la interfaz de usuario de la lógica de la aplicación y proporcionar la comunicación entre los mismos con CORBA, esta tarea se puede realizar de manera semiautomática como veremos en la siguiente sección. Al estructurar la aplicación en tres capas (cliente, lógica de negocio y servidor) utilizando CORBA se reaprovecha todo el código ya existente escrito en COBOL. Se consiguen todas las ventajas que proporcionan CORBA y la arquitectura en tres capas. El mantenimiento de la aplicación resultará mucho menos costoso ya que las capas son independientes entre sí y si necesitamos programar una nueva interfaz de usuario, ésta se podrá realizar incluso en otro lenguaje diferente de COBOL.*

Otra de las características de CORBA es la **invocación dinámica**, la llamada a objetos no tiene por qué ser estática (programada de antemano en el cliente), sino que un cliente puede buscar que servidores y que servicios están disponibles y llamarlos de forma dinámica durante la ejecución.

**Escalar** un sistema es una tarea fácil, por ejemplo si se tienen varios agentes, cosa que es completamente posible, no hay que preocuparse por balancear la carga ya

que hay implementaciones de CORBA, por ejemplo Smart Agent de Visibroker, que lo hacen automáticamente. Así si el número de clientes aumenta, sólo habrá que introducir una nueva máquina con otro agente y CORBA se encargará de repartir el trabajo entre todos los agentes.

Además, aumenta la **tolerancia a fallos** ya que si se tienen varios agentes y uno de ellos cae, automáticamente todos los clientes que estén conectados a él pasan a otro agente, de forma que el cliente no notará nada (sólo un pequeño retardo).

CORBA ofrece la posibilidad de **encriptar** la información evitando los problemas de seguridad al enviar la información de una capa a otra. Pero se nos pueden presentar otros problemas de seguridad derivados de la arquitectura de CORBA. Si la capa lógica, al acceder a un determinado agente, hace una comprobación de usuario y contraseña y nuestro agente se cae, CORBA dará servicio a nuestro cliente con otro agente que esté disponible (si existen otros agentes). Puede suceder que el nuevo agente que se encargara de atendernos, no nos permita el acceso, pero nos habremos saltado la comprobación. Este aspecto puede ser tratado manejando la excepción que se lanza cuando cae el agente.

## 8. CONCLUSIONES

En este trabajo hemos dado un repaso en cierta profundidad a la arquitectura de desarrollo de aplicaciones en tres capas, una nueva forma de construir *software* que aboga por una completa separación entre la lógica del programa, las interfaces de usuario y el sistema de almacenamiento de datos.

Creemos que esta arquitectura permite construir aplicaciones reutilizables, escalables y fáciles de adaptar a las necesidades de clientes muy diferentes, pero también presenta algunos inconvenientes derivados, en su mayor parte, de la relativa inmadurez de las herramientas que soportan esta forma de construir aplicaciones. Debido a esta y otras razones, algunos autores se preguntan si realmente es aconsejable que las empresas acometan reto de empezar a desarrollar aplicaciones en tres capas en vez de continuar utilizando el modelo clásico cliente/servidor que tan buenos resultados ha dado. Quizás el trabajo más crítico en este aspecto es el de [Fryer, 1995]. En nuestra opinión, el desarrollo de nuevas aplicaciones utilizando este modelo reportará en un futuro inmediato tan buenos resultados como los que proporciona la arquitectura cliente/servidor clásica y abrirá a las empresas enormes posibilidades en el mundo de Internet y en el que CORBA jugará sin duda alguna un papel protagonista.

## 9. AGRADECIMIENTOS

No nos gustaría dejar pasar la ocasión sin expresar nuestro agradecimiento a. José R. Salgado, Juan M. Macías y Beatriz Bernárdez por la ayuda prestada para la elaboración de este trabajo.

## 10. REFERENCIAS BIBLIOGRÁFICAS


[Bass *et al.*, 1991] ***Developing Software for the User Interface.*** *L. Bass y J. Cuotaz.* Addisson-Wesley. 1991.

[Blaha *et al.*, 1997] ***Object-Oriented Modeling and Design for Database Applications***. *M. Blaha y W. Premerlani*. Prentice Hall. 1997.

[Cornelious., 1998] ***Using CORBA and JDBC to Produce Three-Tier Systems.*** *B. Cornelious.* ACM SIGPLAN Notices, 33(4). 1998

[Edelstein, 1994] ***Unraveling Client/Server Architecture.*** *H. Edelstein.* DBMS 34(7). 1994.

[Fryer, 1995] ***Three-Tier Client-Server Computing?.*** *B. Fryer.* Open Computing, 12(3). 1995.

[Gallaugher *et al.*, 1996] ***Choosing a Client/Server Architecture. A Comparison of Two-Tier and Three-Tier Systems.*** *J. Gallaugher y S. Ramanathan.* Information Systems Management Magazine, 13(2). 1996.

[Inprise, 1999] ***Delphi Home Page.*** www.borland.com/delphi

[MS1, 1999]***Microsoft COM and DCOM.*** www.microsoft.com/com

[MS2, 1998] ***Mastering Distributed Application Design.*** Microsoft Press. 1998.

[Nassif *et al.*, 1993] ***Issues and Approaches for Migration/Cohabitation between Legacy and New Systems.*** *R. Nassif y D. Mitchusson.* SIGMOD Record, 22(2). 1993.

[Noffsinger *et al.*, 1998] ***Legacy Object Modeling Speeds Software Integration***. *W.B. Noffsinger et al.* Communications of the ACM, 41(12). 1998.

[OMG, 1999]***Object Management Group Home Page.*** www.omg.org

[Oracle, 1999] ***Oracle Home Page.*** www.oracle.com

[Peña, 2000] ***Problemas de Interoperatividad en CORBA.*** J. Peña, R, Corchuelo, A. Ruiz, F. Ferrer. Jornadas Tecnológicas de Cádiz. 2000.

[Schoffner, 1997] ***Java Step by Step: Scale an Application from Two to Three Tiers with JDBC.*** *M. Shoffner*. JavaWorld: IDG's magazine for the Java community, 2(6). 1997.

[Schussel, 1999] ***Client/Server: Past, Present, and Future***. *G. Schussel.* www.dciexpo.com/geos/

[Softis, 1999] ***Softis Home Page.*** www.softis.com

[Xerox1, 1999] ***Aspect Oriented Programming.*** www.parc.xerox.com/spl/projects/aop/

[Xerox2, 1999] ***Aspect Home Page.*** www.parc.xerox.com/spl/projects/aop/aspectj/