UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Escuela Nacional de Estudios Profesionales
ACATLÁN

"SERVIDOR DE COMPONENTES JAGUAR CTS"

T E S I N A

QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

P R E S E N T A :

OLIVIA ERNESTINA SÁNCHEZ GRAILLET

Asesor: Lic. Guadalupe del C. Rodríguez Moreno

Acatlán, Edo. de México
Octubre del 2001

Agradecimientos

A Dios,

A mis padres Rebeca y Avelino,

A mis hermanos Rebeca, Luis y Laura,

A mis abuelitos Olivier, Maclovia, Olga y Avelino,

A mis amigas y amigos,

A mis profesores,

A mi Universidad.

Les dedico este trabajo, pues sin ustedes no lo hubiera podido hacer.

Con amor.

Olivia.

Contenido

Introducción

La arquitectura de cómputo ha evolucionado desde los sistemas de archivos, pasando por el enfoque de dos capas, hasta llegar al de tres capas, la cual consiste de un cliente, que presenta la información al usuario; una capa intermedia, que maneja la lógica de negocios; y de un servidor de datos, en donde éstos son almacenados. A diferencia del modelo de dos capas se ha separado la lógica para poder ser fácilmente localizada y reutilizada.

Debido a la tendencia de utilizar sistemas de tres capas, especialmente en sistemas transaccionales y de consulta para muchos usuarios, las herramientas que facilitan el desarrollo de este tipo de aplicaciones y que cumplen con los estándares internacionales, están teniendo auge.

El Servidor de Transacciones de Componentes Jaguar CTS, es una herramienta adecuada para el ambiente de desarrollo en Sybase, que cumple con estándares internacionales como CORBA y COM++.

Esta tesina pretende dar a conocer la herramienta que es Jaguar, para ser tomada en cuenta como una opción para el desarrollo de sistemas de tres capas. Está dirigida a personas interesadas en desarrollar este tipo de aplicaciones en servidores de transacciones en sistemas que se ejecuten o no en la Internet.

La Web o World Wide Web es una arquitectura computacional distribuida y es uno de los servicios que proporcionan los servidores de Internet. Mediante ésta, se logra navegar fácilmente con una interfaz gráfica a través de la red a diferentes direcciones. Por lo cual, el impacto que ha tenido la Web en la sociedad ha sido que cualquier persona, no importando edad o nivel de estudios, tenga acceso a numerosas fuentes de información en todo el mundo.

Por otro lado, la Internet y su potencial han redefinido el modo en que los negocios operan y la forma de construir sistemas de cómputo. Para que la red no sólo sirva para presentar publicidad, se debe definir la manera de realizar la reingeniería para que una infraestructura diseñada originalmente para presentar textos e imágenes estáticos, se transforme en una arquitectura robusta, capaz de soportar procesos de transacciones a gran escala de forma segura.

En la búsqueda de combinar una Web que cuente con acceso y despliegue universales, con la automatización de distintos tipos de procesos; en el presente trabajo se examinará la propuesta de Jaguar CTS de Sybase, para alcanzar este fin.

En el primer capítulo se describe el modelo de tres capas, sus antecedentes, sus ventajas y desventajas. Se explica el papel de Jaguar CTS en este esquema y se hace una comparación entre algunos servidores de aplicaciones.

Debido a que Jaguar es un servidor de transacciones orientado a componentes, en el segundo capítulo se define el concepto de componente y se presentan los tipos de componentes que se manejan en Jaguar CTS, su diseño, sus propiedades y su construcción.

En el tercer capítulo se explica el marco de trabajo que Jaguar proporciona para desarrollar la lógica de la capa intermedia de aplicaciones distribuidas basadas en componentes, y como proporciona soporte para la construcción de la administración del ciclo de éstos, del manejo de transacciones e hilos, y de la seguridad. Este capítulo es de gran importancia, ya que en él se muestra el funcionamiento de Jaguar.

En el cuarto capítulo el enfoque está dirigido a aplicaciones en la Web, por lo cual se explican conceptos referentes a ésta. Se describe la forma de acceder a los componentes Jaguar desde un applet de Java, así como el uso de los servlets de Java en Jaguar.

En el último capítulo se explican los pasos para crear aplicaciones que utilicen Jaguar, se mencionan algunos requerimientos de sistemas que podrían utilizar la técnica de tres capas, se presenta una aplicación ilustrativa del uso de Jaguar, y finalmente, se describen casos de organizaciones internacionales que lo han utilizado. La aplicación de ejemplo es de la empresa Ericsson, la cual fue escogida debido a que es una compañía a nivel mundial que ya ha tenido experiencia con Jaguar para algunos de sus desarrollos.

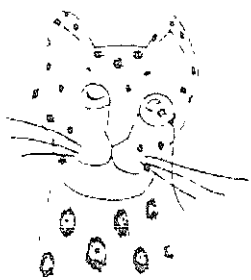
La investigación se basó en Jaguar CTS 3.5 de Sybase como servidor de componentes, siendo esta versión la última en el mercado al momento de la realización de este trabajo.

La mayor fuente de información para este tema ha sido Internet, ya que de ésta se puede obtener la información más reciente, además de que no existe suficiente bibliografía de otro tipo, sobre dicho tema.

Así mismo se puede obtener fácilmente información de empresas de otras partes del mundo, que han utilizado la tecnología de Jaguar CTS, y de esta forma poder compartir sus experiencias.

"Lo último que se sabe cuando
se realiza un trabajo,
es por donde empezar."

Blaise Pascal.



Capítulo I Servidor Jaguar CTS en el Modelo de Tres Capas

La arquitectura de una aplicación es la vista conceptual de la estructura de ésta. Toda aplicación contiene código de presentación, de procesamiento y de almacenaciento de datos. Las arquitecturas difieren según la distribución de su código.

I.1 Arquitectura de Tres Capas

Actualmente, el movimiento hacia arquitecturas distribuidas de tres capas es muy fuerte y ha sido impulsado por el aumento de aplicaciones Web. Este tipo de arquitecturas posibilitan que los elementos de presentación, lógica del negocio y los datos de la aplicación se encuentren claramente separados y se ejecuten en diferentes máquinas conectadas en un entorno de red.

Lo que hace interesantes a las arquitecturas de tres capas en un entorno Web, es que las aplicaciones pueden utilizar interfases de usuario basadas en un navegador que acceden a la lógica del negocio y a componentes de datos ubicados de forma segura detrás de los firewalls corporativos. El enfoque de dos capas no se ha adaptado bien a la Web, puesto que los datos y la lógica de la aplicación quedan expuestos, de forma que pueden ser manipulados por el usuario. Por otra parte, con el enfoque de tres capas se obtiene mayor escalabilidad, seguridad y reutilización del código en las aplicaciones convencionales, como se explicará más adelante.

Para explicar el papel que tiene Jaguar en la capa intermedia de un modelo de tres capas, primero se describirán sus antecedentes y modalidades, y finalmente la participación de Jaguar en esta capa.

I.1.1 Antecedentes de la arquitectura de tres capas

I.1.1.1 Servidor de Archivos

Antes de la utilización de las computadoras personales dentro de una red, se usaban los mainframes, los cuales eran sistemas grandes, rápidos y con un alto costo, capaces de controlar a cientos de usuarios simultáneamente, así como a cientos de dispositivos de entrada y salida. Contaban con terminales "tontas", que sólo presentaban la información al usuario, sin realizar ningún tipo de lógica ni almacenamiento de datos. Actualmente todavía son utilizados, ya que soportan varios programas simultáneamente, aunque tienen las desventajas de su costo y tamaño.

Las primeras redes de computadoras personales se basaban en el principio de archivos

compartidos. En este esquema el servidor simplemente descarga o transfiere archivos desde un área compartida hasta una terminal, en la cual se realiza la lógica del sistema. Este enfoque se dio a conocer por productos del estilo Xbase (multi-usuarios) como dBASE, FoxPro y Clipper. Es simple y funciona bien mientras haya pocos accesos simultáneos, la información no se actualice frecuentemente y el volumen de datos sea poco comparado con una red de área local (LAN).

I.1.1.2 Arquitectura de dos capas

Este modelo cuenta con un verdadero servidor de base de datos que satisface una solicitud con sólo una respuesta, en lugar de enviar un archivo completo.

Esta arquitectura redujo significativamente el tráfico en la red e hizo posible que la información actualizada estuviera disponible para muchos usuarios en una red de área local.

Sus ventajas son :

- Procesamiento de transacciones para más usuarios.
- Tareas bien definidas.
- Aplicaciones que funcionan durante un tiempo largo.
- El tamaño de la aplicación puede ser muy grande: divisional o empresarial.
- Mantenimiento de código plano.

Sus desventajas son :

- Pruebas cada vez que se hace algún cambio en el código.
- Administración de las máquinas cliente.
- Un cambio representa varios ajustes en diferentes partes de la aplicación.
- Manejo manual del funcionamiento.
- Influencia del hardware existente.
- Control descentralizado.
- Sobrecarga de trabajo en el cliente.
- Cuellos de botella en el servidor de base de datos con muchos usuarios y conexiones simultáneas.

- Lógica del negocio almacenada en una base de datos escrita en lenguaje propietario.

Un buen uso de esta arquitectura, es la implementación de un cliente delgado y un servidor grueso que realice la lógica del negocio.

Un error común al desarrollar un sistema, es crear un prototipo con una arquitectura de dos capas y después aumentar el número de usuarios que acceden al servidor. Generalmente ésto se convierte en un sistema ineficaz, ya que el servidor se sobrecarga. Para escalar apropiadamente a un mayor número de usuarios, generalmente es necesario cambiar a una arquitectura de tres capas.

I.1.2 Definición de la arquitectura de tres capas

Una arquitectura de tres capas introduce un "agente" entre el cliente y el servidor, que puede realizar funciones como:

- Servicios de traducción: adaptar una aplicación heredada de un sistema *mainframe* a un ambiente cliente/servidor.

- Medición de servicios: monitoreo de transacciones. Puede limitar el número de solicitudes simultáneas a un servidor.

- Servicios inteligentes: dirigir una solicitud a varios servidores, compilar los resultados y regresarlos como una sola respuesta al cliente.

Este tipo de arquitectura consiste de las siguientes partes:

- **Capa cliente:** contiene la lógica de presentación de la información en forma gráfica e interactúa con el usuario, al permitirle capturar datos y hacer consultas.

- **Capa intermedia:** está situada entre el cliente y los servidores de bases de datos. Puede realizar diferentes funciones como encolamiento de tareas, ejecución de aplicaciones, funciones de programación, prioridades de los procesos y protección de los datos para no ser accedidos directamente por el cliente. En esta capa, los procedimientos *síncronos*[1] del modelo de dos capas se convierten en *asíncronos*; es decir, el cliente puede enviar sus solicitudes a la capa intermedia, desentenderse, y tener la seguridad de que obtendrá una respuesta apropiada en poco tiempo.

- **Capa de acceso a datos:** es la responsable del almacenamiento de datos. Contiene la lógica de la interfaz con un sistema de almacenamiento de datos, como

[1] Las palabras cuyo significado no sea claro, pueden ser consultadas en el Glosario.

puede ser un sistema de base de datos, un sistema jerárquico de archivos, o cualquier otro tipo de fuente de datos externa.

Los límites entre estas capas son lógicos, por lo que es posible tenerlas en una misma máquina física, siempre y cuando el sistema esté bien estructurado y sus límites bien definidos. Así mismo, esta arquitectura permite establecer fácilmente los límites de las reglas de presentación, del negocio y del acceso a datos.

Aunque las capas del sistema sean lógicas y estén o no en distintas máquinas físicas, los clientes generalmente se encuentran en su propia máquina, por lo que cuando hacen solicitudes al servidor de aplicaciones, no acceden directamente a los datos.

Para acceder a los datos en este tipo de arquitectura, se puede contar con varios servidores. Cuando los componentes son desplegados a través de más de una máquina, el término "N-capas" se puede emplear. Sin embargo, no es adecuado, ya que, aunque la capa intermedia estuviera repartida en diez aplicaciones que se encuentren en seis máquinas distintas, el sistema seguiría formado sólo por tres capas.

En la siguiente figura se ilustra la división en tres capas y el contenido de cada capa.

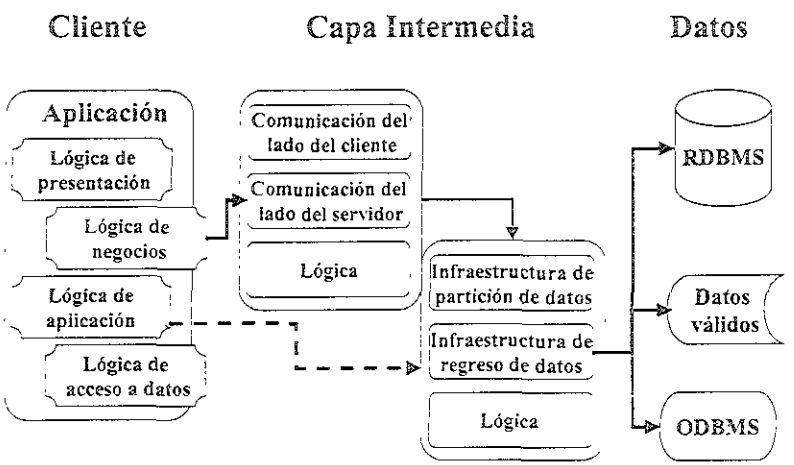


Fig. I.1 División en tres capas

Una arquitectura de tres capas reduce el tráfico en la red y mejora su funcionamiento. El cliente interactúa con la capa intermedia a través de un protocolo estándar, como lo hacen las librerías dinámicas (DLL), las interfases de aplicación (API) o las llamadas a procedimientos remotos (RPC).

La capa intermedia interactúa con el servidor a través de protocolos estándares de bases de datos. Contiene la mayor parte de la lógica de la aplicación, traduciendo las llamadas del cliente en solicitudes a la base de datos, y los datos de regreso en datos para el cliente.

Si la capa intermedia estuviera localizada en la misma máquina que la base de datos, éstas podrían mantenerse unidas por una interfaz hecha con un lenguaje de tercera generación, lo cual resultaría en una interacción controlada y de mayor desempeño, evitaría un procesamiento costoso, y se aprovecharía mejor la red. Más aún, la capa intermedia podría ser distribuida a una tercer máquina para mejorar el procesamiento.

El problema de instalación de nuevas aplicaciones y de distribución de cargas de procesamiento que se tenían en una estructura de dos capas, desaparecen en este modelo.

Al aumentar el número de usuarios en línea, el funcionamiento de una arquitectura de dos capas empieza a fallar debido al proceso de conexión al servidor del *DBMS*, que mantiene un hilo [2]para cada cliente conectado al servidor de forma que, incluso cuando no se está realizando ningún trabajo, el cliente y el servidor siguen intercambiando mensajes para comprobar que todavía hay conexión de forma continua entre ellos. Si hubiera algún problema con la conexión, el cliente tendría que realizar un proceso para reiniciar la sesión. Con 50 clientes y el hardware de las computadoras personales actuales, esto no es problema. Pero cuando se tienen 2,000 clientes en un solo servidor, el funcionamiento se vuelve inadecuado. La arquitectura de tres capas es apropiada para ambientes empresariales.

Las aplicaciones se pueden escribir en lenguajes estándares en lugar de lenguajes propietarios o de procedimientos almacenados limitados. El lenguaje de datos utilizado para implementar procedimientos en servidores tipo SQL es propiedad de cada proveedor, como Oracle, Sybase, Informix e IBM, que utilizan diferentes variantes del lenguaje SQL. El enfoque de lenguajes propietarios es bueno desde el punto de vista del funcionamiento, pero tiene la desventaja de no permitir escoger el DBMS para las aplicaciones.

A diferencia del enfoque de dos capas, en el que las aplicaciones implementadas no son fáciles de separar, ni de cambiar alguna de sus funcionalidades de un servidor a otro, pues esto puede requerir regenerar el código; en el enfoque de tres capas se pueden colocar módulos de código de la aplicación en varias computadoras.

El precio de esta flexibilidad y buen funcionamiento, es un ambiente de desarrollo más difícil de usar que el de las aplicaciones de dos capas.

---

[2] El término de hilo se trata con más detalle en el capítulo IIII.4 1

I.1.3 Requerimientos

Un elemento necesario en una arquitectura de tres capas es una infraestructura distribuida de cómputo que proporcione los servicios para que los componentes de la aplicación se mantengan transparentemente distribuidos en varias máquinas físicas. Así mismo, que permita reconfigurar el ambiente de aplicaciones distribuidas para adecuarse a mayores cargas de trabajo y a fallas inesperadas en el hardware en el momento en que se presenten y sin necesidad de suspender el servicio.

Los principales servicios que proporciona una infraestructura distribuida son:

- **Directorio de Servicios:** mantiene una lista dinámica de todos los servicios de la aplicación. En el momento en que un cliente hace una solicitud, el directorio de servicios localiza algún servicio que la pueda atender y que le de instrucciones al cliente para comunicarse con él.

- **Servicio de Seguridad:** lleva registro de todos los usuarios autorizados y de las operaciones permitidas a cada usuario o grupo de usuarios; proporciona un solo servicio de entrada para todos los sistemas de la empresa, de modo que si la autenticidad de un usuario ha sido aprobada por el sistema de seguridad, ese usuario podrá acceder a todos los sistemas.

- **Servicio de Administración de Aplicaciones:** mantiene una configuración dinámica de los servicios de la aplicación. Es responsable de iniciar los servicios apropiados de la aplicación en las máquinas correctas y de monitorear dichos servicios, asegurando que estén disponibles y funcionando dentro de los parámetros requeridos.

I.1.4 Seguridad

Para que un sistema tenga éxito, debe ser seguro y no permitir que cualquier usuario acceda a los datos. Para lograrlo, se debe buscar la independencia del ambiente, la seguridad en el acceso a cada sistema y en toda la empresa simultáneamente.
Los sistemas de seguridad de tres capas cuentan con un control centralizado y autentificado por un servicio, y controlan el acceso a nivel cliente, de modo que los usuarios no puedan acceder directamente a los datos.

Los modelos de tres capas requieren de la llamada "tecnología habilitadora"; que consta de dos capas. La inferior que contiene servicios elementales de plataformas y seguridad; y la superior que incluye soporte para los sistemas cliente/servidor empresariales, como balanceo de cargas, e integración de *mainframe* y transacciones.

I.1.5 Ventajas

Una arquitectura de tres capas presenta ventajas en las áreas de control, confiabilidad, escalabilidad, funcionamiento, flexibilidad de crecimiento y cambio, estándares y costos. continuación se explicarán cada una de estas áreas.

1. CONTROL

A diferencia del modelo de dos capas, en el que sólo los datos pueden ser accedidos por todos los desarrolladores, los componentes hacen que la lógica del negocio y los servicios estén disponibles a través de la red. Por ejemplo, un número de inventario tiene un dígito verificador y el cálculo de ese dígito puede estar disponible en el servidor, para ser usado por otros desarrolladores.

La ejecución de procedimientos en el servidor es segura y está bajo control, de forma que pueden ser probados, delimitados y auditados. Los servicios son construidos de forma encapsulada con una interfaz conocida y con una semántica pública, de forma que sean bien conocidos y limitados; y que la base de datos de la aplicación no esté expuesta a un acceso directo.

Los servicios encapsulados sirven como un *firewall* entre los datos de la aplicación y los programas de la interfaz de usuario, lo cual mejora la confiabilidad de la aplicación.

Puesto que la lógica del negocio está en el servidor, la distribución del software se facilita, ya que es más sencillo actualizar los programas en un servidor de aplicaciones que en muchas máquinas cliente. Así mismo, cuando el código de la interfaz de usuario se modifica, necesita ser redistribuido, para lo cual el servidor de aplicaciones se puede usar como punto de distribución.

2. CONFIABILIDAD

En caso de producirse un cuello de botella en la red, el proceso del servidor puede cambiarse a otros servidores en tiempo de ejecución, lo cual se conoce como balanceo dinámico de cargas.

La lógica de la aplicación no está unida directamente a la estructura de la base de datos o a un DBMS particular. Los objetos de acceso a datos son los únicos componentes de la aplicación que interactúan directamente con la base de datos, lo cual proporciona mayor flexibilidad para hacer cambios tecnológicos y comerciales.

El hardware de un servidor de aplicaciones es más confiable que el de una computadora personal, ya que se ubica en un ambiente seguro, como un centro de cómputo.

Los servicios más importantes pueden ser replicados en varios servidores, lo cual les da disponibilidad. La capa intermedia puede dirigir las transacciones hacia varias instancias basándose en la cargas de trabajo y en la disponibilidad.

Una aplicación cliente/servidor con procesamiento de transacciones bien diseñada, consiste de transacciones conocidas, claramente demarcadas y controladas; y opcionalmente de un monitor de transacciones para garantizar la integridad de éstas a través de distintas plataformas.

Generalmente el control de las transacciones es implementado en un servicio de control que residente en el servidor de aplicaciones. Cuando se emplea un monitor de transacciones, éstas se puede controlar desde una terminal. Con el diseño basado en módulos, que es demandado por la propia arquitectura, se obtiene mayor confiabilidad e implementaciones sin sorpresas.

3. ESCALABILIDAD

Los sistemas de tres capas solucionan los problemas que tenían los de dos capas. El funcionamiento mejora, puesto que los clientes hacen pequeñas solicitudes de servicios en vez de grandes solicitudes de datos, lo cual significa que un menor volumen de datos es enviado por la red. También mejora debido a que los servidores de aplicaciones se pueden ejecutar en muchas máquinas simultáneamente, lo cual ayuda a repartir las solicitudes de los clientes en muchas máquinas, permitiendo una disponibilidad mayor del sistema, un mejor funcionamiento y una mayor escalabilidad. El número de conexiones a una base de datos también se puede reducir, ya que las sesiones de los clientes son manejadas por los servidores de aplicaciones, lo cual hace posible que las conexiones sean compartidas por muchos clientes.

Debido a que el número de instancias en el servidor se pueden controlar y está disponible el balanceo de cargas, se puede evitar más fácilmente la saturación del servidor y se consigue que los problemas de funcionamiento se vayan dando *gradualmente y no de una forma determinante. (La carga en el servidor es conocida* y no aumenta de forma incontrolable al haber más usuarios en línea).

La escalabilidad es el resultado de un buen funcionamiento, así como de otros factores, en particular de:

- Disponibilidad de plataformas de hardware ampliamente escalables.
- Capacidad de añadir otro servidor, para un propósito específico, en cualquier momento.
- Especialización de los servidores (base de datos, aplicación, comunicaciones, etc.).
- La capa intermedia facilita la escalabilidad al hacer posible la utilización de varios servicios de forma productiva.

4. FLEXIBILIDAD

Las aplicaciones crecen y cambian, por lo que la escalabilidad del diseño y la modularidad son importantes, así como poderlas expandir rápidamente.

Se puede trabajar simultáneamente en las tres capas. Las interfases fundamentales son bien conocidas y sirven de base para los sub-equipos que trabajan en cada capa.

A continuación se analiza la flexibilidad de cada capa.

- **Capa Cliente:** Sólo cambia la interfaz de usuario mientras que los servicios de negocios y de datos permanecen iguales. Las interfases pueden operar simultáneamente, ya que la base de datos es invisible para los clientes. Al agregar nuevas aplicaciones, éstas se pueden integrar fácilmente en una sola vista para el cliente. Los servicios de la segunda y la tercer capa son desarrollados y probados independientemente.

- **Capa de servicios de aplicación:** La segunda capa es mucho más estable que la primera. Las reglas del negocio no cambian tan rápido como la interfaz de usuario. Si algo cambia, un nuevo servicio encapsulado puede reemplazar al anterior sin mayor problema; los nuevos servicios de aplicación pueden añadirse en cualquier momento y las nuevas aplicaciones pueden conectarse a clientes que ya existen.

- **Capa de servicios de datos:** Al igual que con los servicios de aplicación, la modularidad natural de esta arquitectura permite modificar y agregar cómodamente servicios de datos. También se puede cambiar gradualmente a un nuevo DBMS, usar DBMS's de diferentes proveedores al mismo tiempo y no necesitar un *gateway*. Además, no todos los datos se almacenan en bases de datos relacionales. El servidor es completamente programable y se pueden integrar archivos secuenciales y de índices, bases de datos no relacionales, servidores de fax y salida para EDI y otras redes.

5. ESTÁNDARES

Escoger la herramienta de desarrollo para la aplicación cliente es difícil y la posibilidad de que siga vigente de 5 a 10 años es mínima. Los servicios de aplicaciones pueden, y de preferencia deben ser escritos en lenguajes estándares como COBOL o C/C++. El uso de lenguajes propietarios (como los procedimientos almacenados) para escribir la lógica del negocio es innecesario y peligroso.

Los servicios de datos deben utilizar lenguajes estándares en los códigos para transmitir la información a la aplicación. Se debe procurar usar moderadamente procedimientos almacenados, lo que trae como resultado poder reemplazar a un DBMS por otro, en caso de ser necesario.

Un monitor de transacciones puede utilizarse para coordinar las transacciones a lo

largo de múltiples procedimientos, máquinas, y/o bases de datos heterogéneas.

6. COSTOS

Con el avance rápido de la tecnología y el decremento de los costos, muchos negocios pueden adquirir un servidor de tres capas y paquetes de software para estas arquitecturas. El costo de mantenimiento es otro punto a considerar. Debido al incremento de la escalabilidad, confiabilidad y seguridad, los sistemas de tres capas son más recomendables a largo plazo, porque aunque el costo inicial pudiera ser elevado en dinero y esfuerzo, pero actualizarlo sería menos costoso.

En un sistema de tres capas, varias aplicaciones pueden estar operando simultáneamente, por lo que para integrar una nueva aplicación, sólo se necesita instalar el nuevo software y transferirle los datos. Así mismo, la instalación de nuevo hardware sólo requeriría de conectarlo a la máquina y configurarlo para el servidor.

La confiabilidad de los sistemas de tres capas, contribuye a la reducción de sus costos. Por ejemplo, en caso de presentarse una falla o un problema en el hardware, éste se puede arreglar fácilmente sin tener que apagar por completo el servidor.

La seguridad es otro factor que ayuda a mantener los costos bajos en los sistemas de tres capas. Por ejemplo, debido a que los clientes no tienen acceso directo a los datos, la posibilidad de un acceso no deseado se reduce, lo que a su vez significa una rebaja en los costos, ya que el personal de mantenimiento del servidor no tiene que estar arreglando las fallas. Además, si algo ocurriera, no se tendría que dar de baja al servidor.

I.2 Opciones de capas intermedias

I.2.1 Monitores de procesamiento de transacciones

El tipo más básico de capa intermedia es el monitor de procesamiento de transacciones o monitor TP. Puede pensarse en él como un tipo de servicio de mensajes encolados. El cliente se conecta al monitor TP en lugar de al servidor de bases de datos. La transacción es aceptada por el monitor, que la envía a la cola y supervisa que se complete correctamente.

Los monitores TP tuvieron su auge con los mainframes de los años setentas, al accederlos en línea por el método de tiempo compartido o por el procesamiento de transacciones en línea (OLTP). El primero se utilizaba para desarrollar programas y apartaba los recursos de las computadoras con un simple algoritmo calendarizador; mientras que la programación *OLTP* era más sofisticada y era atendida de acuerdo a prioridades. Los monitores *TP* generalmente se usaban en el ambiente OLTP, siendo el

más popular *CICS* de IBM.

En los años noventas, las aplicaciones cliente/servidor se popularizaron y el uso de los monitores TP disminuyó. Esto sucedió porque muchos de los servicios proporcionados por un monitor TP se encontraban disponibles como parte de un DBMS o como software de capa intermedia, proporcionados por vendedores como Sybase, Gupta y Oracle.

En la siguiente figura se muestra el esquema de una red que utiliza un monitor TP.

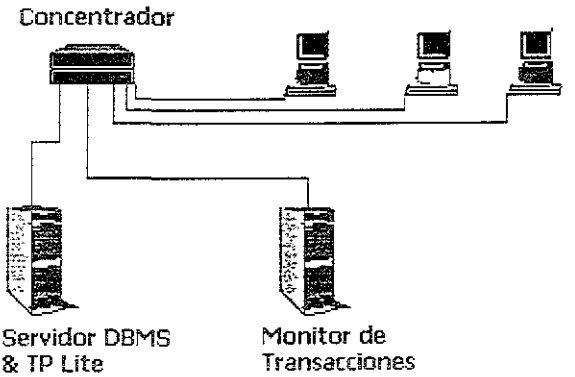


Fig. I.2 Monitor de Procesamiento de Transacciones

Los monitores TP se han vuelto a utilizar debido a que sus mecanismos de colas proporcionan el efecto de embudo (funneling) que reduce el número de hilos que un servidor DBMS necesita mantener. El cliente se conecta al monitor, el cual acepta el mensaje y lo pone en cola para procesarlo en la base de datos. Una vez que el monitor ha aceptado el mensaje, el cliente puede ser liberado para atender el siguiente mensaje. Entonces, la sesión síncrona basada en una arquitectura de dos capas se convierte en asíncrona debido al monitor TP. El monitor también ayuda a acceder el servidor de bases de datos.

Algunos servicios importantes que proporciona un monitor TP son:

- Capacidad para actualizar diferentes DBMS en una sola transacción.
- Conectividad con distintas fuentes de datos, incluyendo archivos planos, DBMS no relacionales y mainframes.
- Capacidad para dar prioridades a las transacciones.

- Buen esquema de seguridad.

El uso de una arquitectura cliente/servidor de tres capas con un monitor TP da como resultado un ambiente más escalable que el de un enfoque de dos capas con conexiones directas entre cliente y servidor. Para aplicaciones realmente grandes (1,000 usuarios) los monitores TP son una de las soluciones más efectivas.

I.2.2 Servidor de Mensajes

Otra tecnología para implementar la computación en tres capas es el envío de mensajes, disponible por parte de compañías como IBM, Sybase y Oracle. Un servidor de mensajes puede verse como un tipo de monitor TP de segunda generación que proporciona el mismo proceso *funneling*. Los mensajes son procesados de forma asíncrona con los niveles de prioridad apropiados, y a semejanza de un monitor TP, proporciona conectividad a fuentes de datos distintas de las *RDBMS*.

Un mensaje es un objeto auto contenido con información sobre sí mismo, como a dónde debe ir y qué es lo que debe hacer al alcanzar su destino. El mensaje tiene al menos dos partes: el encabezado, que contiene la prioridad, dirección y número de identificación; y el cuerpo del mensaje, que contiene la información a ser enviada, la cual puede ser cualquier cosa, incluyendo textos, imágenes o transacciones.

La arquitectura de los servidores de mensajes se diseñó en torno a la inteligencia del mensaje en sí, a diferencia del ambiente de los monitores TP que definen la inteligencia del sistema en el monitor o en la lógica del proceso del servidor de aplicaciones.

En un ambiente que utilice un monitor TP, las transacciones son simples paquetes de datos que viajan a través de una conexión preexistente y predefinida hacia el monitor que analiza y procesa la transacción. La mayoría de las veces remite la solicitud a una aplicación de la capa servidor y en caso de que el monitor TP no entienda los datos, éstos no serán procesados. Finalmente, el monitor TP necesita conocer tanto acerca de la transacción como la capa del servidor.

El servidor de mensajes es sólo un contenedor de mensajes y procedimientos. Las operaciones efectuadas en el mensaje por este servidor, están relacionadas con la comunicación (p. e. encriptar mensajes en un servicio, y desencriptarlos en otro). Los mensajes contienen toda la información necesaria para servicios de red (p. e. direcciones de red lógicas y físicas), y dado que el mensaje es inteligente, la capa intermedia basada en mensajes resulta ser más flexible que un monitor TP.

Para mensajes que contienen información sobre los servicios de red, la capa intermedia puede servir simplemente como punto de enrutamiento entre dos clases de servicios de red. Para los mensajes inteligentes, la capa intermedia puede ejecutar un procedimiento almacenado o una regla de negocios, según lo dicte el mensaje. Esta

abstracción de la capa intermedia, lejos de los contenidos y del comportamiento de la información que fluye a través de ella, convierte al sistema más portátil en diferentes ambientes y redes, ya que los detalles de la transmisión de la información quedan ocultas bajo el servicio de mensajes.

Los sistemas de mensajes tienen un diseño robusto, ya que emplean una lógica de almacenamiento y de envío que asegura la entrega de mensajes a pesar de existir fallas. También proporcionan independencia de tecnologías de enlace como los protocolos alámbricos o los inalámbricos, que no requieren de una conexión constante entre el cliente y el servidor.

La entrega de mensajes puede programase para realizarse después o durante las fallas. Puesto que los sistemas de mensajes soportan la infraestructura inalámbrica, se pueden emplear en el soporte para dispositivos móviles u ocasionalmente conectados.

Una arquitectura común de un servidor de mensajes se vería como en la figura siguiente.

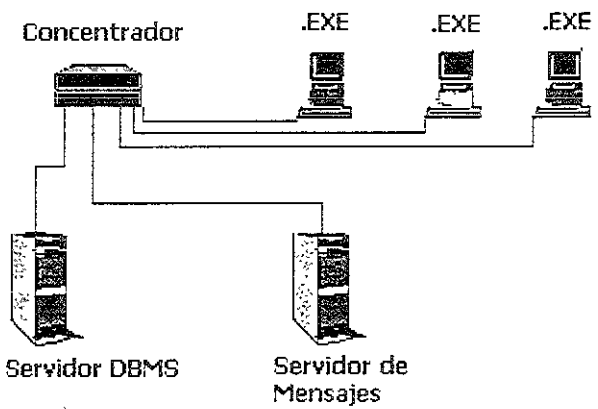


Fig. I.3 Tres capas con un servidor de mensajes

I.2.3 Servidor de Aplicaciones

Generalmente al hablar de una arquitectura de tres capas, se refiere al enfoque de un servidor de aplicaciones, en el que la mayor parte de la lógica del negocio está en un servidor común y compartido, y la computadora personal sólo se usa para los servicios de presentación.

El servidor de aplicaciones no se preocupa por la interfaz de usuario, sino de la lógica

del negocio, su ejecución y la recuperación de los datos. Este servidor opera normalmente bajo un sistema operativo multi-tareas de 32 bits, como NT, OS/2, Netware o UNIX, los cuales trabajan mediante configuraciones de multi-procesamiento simétrico (SMP), que permite que el servidor sea muy escalable en términos de funcionamiento.

Colocar la lógica del negocio en un servidor ofrece las siguientes ventajas para el diseñador de aplicaciones:

- Habiendo menos software en el cliente, exista mayor seguridad, ya que el software importante está en el servidor en un ambiente más controlado.

- Los costos de soporte e instalación del software en un solo servidor son menores que el mantenimiento de muchas computadoras personales.

Con un servidor de aplicaciones en la capa intermedia es más sencillo diseñar una aplicación que utilice un DBMS, ya que sería factible y sencillo cambiar de proveedor de DBMS en un solo servidor que en varias computadoras personales.

- Un servidor de aplicaciones administra eficientemente :

- La instanciación y el ciclo de vida de los componentes

- Las conexiones a bases de datos

- Las transacciones

- La seguridad

La mayoría de las herramientas utilizadas para implementar el enfoque de servidores de aplicaciones en tres capas, ofrecen la posibilidad de dividir las aplicaciones después de haberse construido, lo cual significa que los códigos y los módulos de las funciones pueden reacomodarse en otros servidores, haciendo flexible el funcionamiento.

I.2.4 Servidor de aplicaciones Web

Representa otra opción de capas intermedias. Esta tecnología surgió como un intento de transformar los servidores Web de Netscape y Microsoft en servidores de aplicaciones, actualizando las últimas versiones de sus respectivas interfases de programación de aplicaciones (NSAPI e ISAPI). Los servidores de aplicaciones Web generalmente son hechos a la medida, con una o varias herramientas de desarrollo Web.

Este enfoque proporciona una alta productividad en el desarrollo, pero su desventaja es que la escalabilidad se ve muy limitada debido al uso de una liga directa entre los servidores de aplicaciones y los servidores Web, así como por la falta de protocolos de

comunicación distintos a *HTTP*.

I.2.5 Sistema Manejador de bases de datos orientado a objetos (ODBMS)

Una variante de los servidores de aplicaciones, es el DBMS orientado a objetos (ODBMS) en la capa intermedia, que actúa como acelerador de datos o "hot caché". Es decir, que se puede usar para obtener datos desde un almacén común, para ser utilizados por la aplicación.

Puesto que los tipos de datos extendidos, como video y voz, no están comúnmente soportados por los RDBMS actuales, dichos tipos de datos también podrían ser almacenados en el ODBMS, que podría asociar los datos multimedia apropiados con los datos regresados del RDBMS.

La siguiente figura representa el esquema que utiliza un ODBMS.

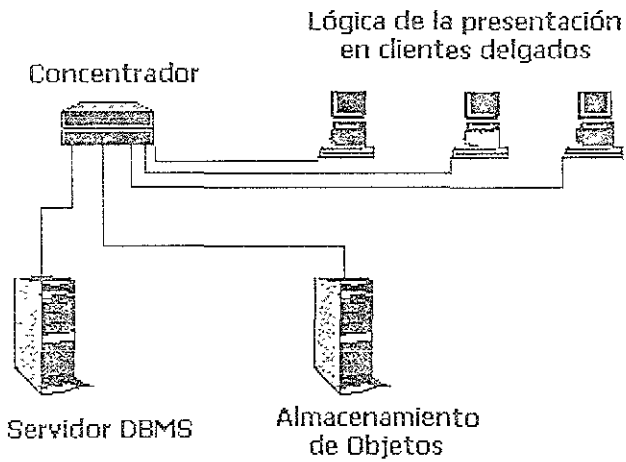


Fig. I.4 Tres capas con un ODBMS

I.2.6 Componentes distribuidos

La implementación de objetos distribuidos en el esquema cliente/servidor cambia la manera de construir las aplicaciones. Una ventaja de este enfoque es que se pueden implementar copias de los objetos en varios servidores. De esa forma, si hubiera alguna falla, sería posible hacer solicitudes a otro servidor.

Con los objetos distribuidos, los cuales contienen dentro de sí mismos los datos y procedimientos, es posible ajustar el funcionamiento de la red con sólo moverlos del hardware sobrecargado a computadoras alternas menos cargadas. Este enfoque es conocido como ajuste por medio de "drag and drop" (arrastrar y soltar).

Una arquitectura de objetos distribuidos también debe ofrecer otros beneficios a los desarrolladores, como los siguientes:

- La misma interfaz puede ser utilizada tanto para construir una aplicación sencilla para una computadora personal, así como para desarrollar una aplicación totalmente distribuida.

- La aplicación puede ser desarrollada y probada localmente y de esta forma tener la seguridad de que trabajará bien cuando sea distribuida.

- Puesto que el desarrollador utiliza un agente de solicitud de objetos (ORB) para la transmisión de servicios, ya no se tiene que preocupar por cuestiones técnicas como colas, tiempos y protocolos.

En la siguiente figura se ilustra el enfoque de tres capas utilizando un ORB.

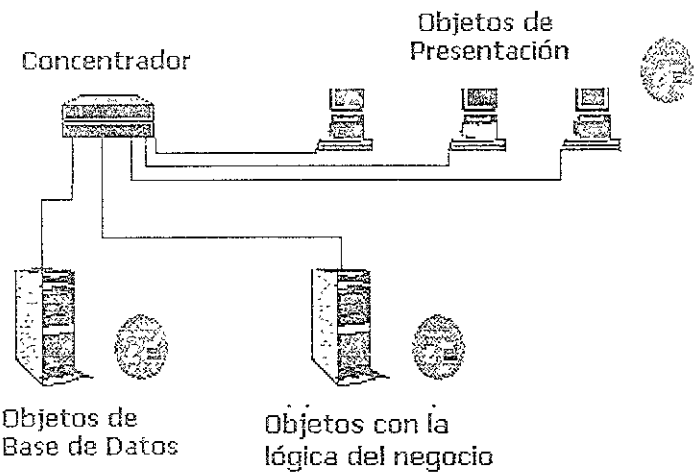


Fig. I.5 Tres capas con ORB

Las principales arquitecturas distribuidas son:

- **CORBA (Common Object Request Broker Architecture).** Es la arquitectura para proyectos distribuidos más importante. Fue creada por el OMG y se basa en los sistemas auto descritos y plantea que la especificación de un servicio debe separarse de su implementación. Principalmente está diseñado para permitir que diferentes componentes interactuen en un canal común de comunicación.

- **COM++ (Component Object Model).** Es una arquitectura de software propietaria de Microsoft, que permite crear aplicaciones a partir de componentes de software y es la base para la creación de servicios de software de alto nivel.

- **CORBA ORBs.** Es la forma en que se comunican entre sí los objetos distribuidos que tienen capacidades multi-capas y de invocación a objetos y a servicios relacionados entre sí. Sin embargo, es compleja y no cuenta con buenas herramientas de soporte. Además, la mayoría de los ORBs tienen mecanismos primitivos de ejecución en el servidor, lo cual limita su funcionamiento y escalabilidad.

I.2.7 Servidor de transacciones

Para cubrir la necesidad de una capa intermedia escalable y fácil de usar, se han creado los servidores de transacciones, los cuales combinan las mejores características de los ORBs y de los monitores de procesamiento de transacciones, con el enfoque basado en componentes, lo que hace posible un desarrollo rápido de aplicaciones escalables para transacciones en línea dentro del Web.

Los primeros servidores de transacciones en estar disponibles fueron :

- Jaguar CTS (Component Transaction Server) de Sybase Inc. Y

- Microsoft Transaction Server (conocido anteriormente como "Viper")

Es de esperarse que aparezcan en el mercado otros productos conforme aumente la demanda de aplicaciones en línea.

A continuación se muestra la combinación de un servidor ORB con un monitor TP, que trae como resultado un servidor de transacciones, que toma las mejores características de dichos servidores, mostradas en el lado derecho del recuadro y desechando las características menos ventajosas, mostradas en el lado izquierdo.

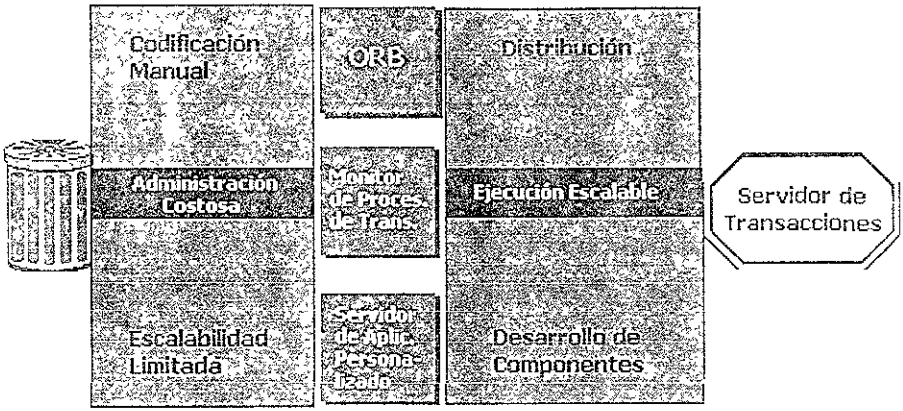


Fig. I.6 Servidor de transacciones: mejores características de los Monitores TP y ORB.

Los servidores de transacciones se definen en función de las siguientes características distintivas:

- Capacidad para manejar transacciones dependientes de la oferta.
- Mecanismo para ejecutar y manejar *servlets*.
- Invocación de objetos distribuidos en aplicaciones multi-capas.
- Soporte para el desarrollo rápido de capas intermedias, incluyendo el desarrollo basado en componentes.

I.2.8 Comparativo de Servidores de Aplicaciones

A continuación se presenta un comparativo de los productos líderes en el mercado, tomando como criterios la facilidad para realizar las siguientes tareas básicas y rutinarias en el uso del servidor de aplicaciones:

- Llamadas asíncronas: siempre que el servidor adquiera un estado o concluya una tarea, lo notifica a uno o más clientes.
- Manejo de transacciones: manejar como una sola transacción la actualización de un archivo plano y de una base de datos.
- Control de concurrencia: garantizar el control de concurrencia al realizar actualizaciones de recursos compartidos.

- Seguridad: autentificar al cliente, garantizando la autorización para solicitar un servicio, y manejar encripción y sumas criptográficas.

- Persistencia de objetos: recuperar y almacenar la instancia de una clase en una base de datos relacional y en archivos planos.

- Interfaz COM: incorporar llamadas a servicios proporcionados por componentes COM.

- Interfaz con aplicaciones legadas: incorporar llamadas a rutinas programadas en C.

- Comunicación entre ORBs: permitir que un objeto acceda a servicios proporcionados por objetos registrados en otro servidor de aplicaciones.

- Balanceo de cargas: verificar la creación automática de nuevas instancias de una clase para distribuir la carga de trabajo.

Los servidores de aplicaciones líderes en el mercado que se comparan son:

- NetDynamics ver 5.0 de Sun Microsystems.

- WebLogic ver 4.5 de BEA Systems Inc.

- WeSphere ver 2.0 de IBM

- Enterprise Application Server (Jaguar) ver 3.5 de Sybase Inc.

La siguiente tabla muestra un comparativo de las características técnicas:

CARACTERISTICAS	JAGUAR	NETDYNAMICS	WEBLOGIC	WEBSPHERE
Plataforma UNIX	X	X	X	X
Plataforma NT	X	X	X	X
Arquitectura CORBA	X	X	-	-
Servicios CORBA	X	-	-	-
Soporte a Objetos Java	X	X	X	X
Soporte a Objetos C++	X	-	-	-
Soporte a Objetos ActiveX	X	-	-	-
Soporte a objetos PB	X	-	-	-
Interfaz gráfica para usuario	X	-	-	-
Conectividad con WebServer	X	X	X	X
Seguridad	X	X	-	-
Herramientas de desarrollo	Enterprise Application Studio(Java, C++, PB)	Netscape Application Builder (Java)	-	WebSphere Studio (Java)
Herramienta de Administración	X	X	X	X
Conectividad con B.D.	X	X	-	-

Tabla I.1 Comparativo de servidores de aplicaciones.

En la tabla anterior se puede ver que Jaguar es el único producto que cumple con el estándar CORBA y que además acepta el mayor tipo de objetos (Java, C++, ActiveX, PB).[3]

[3] La información anterior fue recopilada de las páginas Web de cada uno de los productos evaluados para llenar la ficha técnica y de precios, en noviembre del 2000.

I.3 Jaguar CTS en la capa media

Una tarea difícil es decidir que tecnología utilizar para desarrollar la capa intermedia, para lo cual se tiene que evaluar cual es la que mejor ejecuta y maneja la lógica del negocio.

Los requerimientos principales de esta capa son los siguientes :

- Escalabilidad y capacidad para manejar un gran volumen de usuarios, sesiones, transacciones y conexiones a DBMSs.
- Conectividad de alto desempeño desde los navegadores hasta los almacenes finales de datos.
- Soporte para el desarrollo rápido de aplicaciones Web con procesamiento de transacciones multi-capas en línea.
- Soporte para el manejo de transacciones síncronas y asíncronas.

La capa intermedia interactúa con el servidor de base de datos por medio de protocolos estándares de comunicación y contiene la mayoría de la lógica del negocio, haciendo posible la reutilización del código y el mantenimiento centralizado.

Actualmente las tres alternativas de capas intermedias más utilizadas son:

- Agentes de requisición de objetos (ORBS) de tipo CORBA.
- Monitores de procesamiento de transacciones.
- Servidores de aplicaciones Web.

Cada una de ellas tiene sus puntos fuertes, pero ninguna satisface por completo las necesidades del procesamiento de transacciones en línea en la Web.

Jaguar CTS (Component Transaction Server)

El servidor de transacciones de componentes Jaguar CTS se encuentra dentro del EAServer, el cual es un conjunto integral de servidores de aplicaciones que se usan para desarrollar aplicaciones Web que soportan mucho tráfico, contenido dinámico y procesamiento constante de transacciones en línea.

Además de Jaguar CTS, EAServer consiste de Power Dynamo (servidor de páginas dinámicas), un integrador de aplicaciones y Adaptive Server Anyware (manejador local de bases de datos).

En la arquitectura de N-capas, el cliente interactúa con la capa intermedia (EAS) por

medio de un protocolo estándar de comunicación (IIOP). La capa intermedia (EAS) interactúa con el servidor de base de datos por medio de protocolos estándares de comunicación entre bases de datos (JDBC, ODBC, Open Client) y contiene la mayor parte de la lógica del negocio, traduciendo las llamadas de los clientes en acciones de cálculo, control, y de acceso a bases de datos. Los componentes Jaguar actúan en la capa intermedia, entre las aplicaciones cliente y las bases de datos remotas. Como se muestra en la siguiente figura.

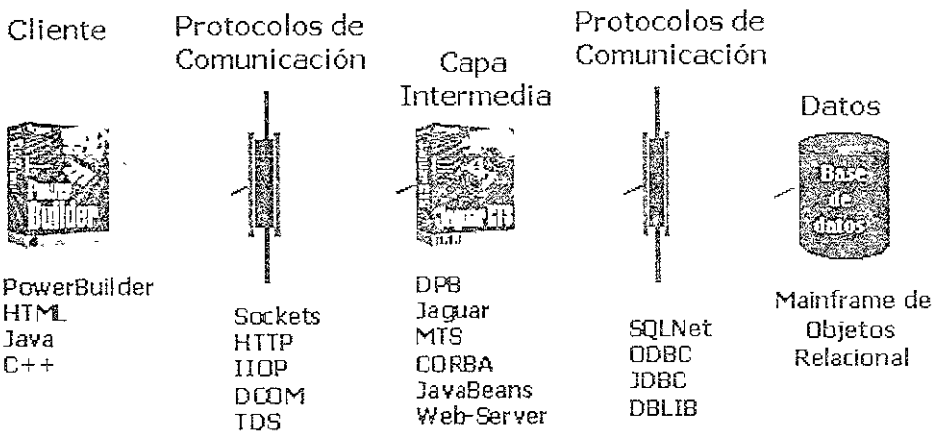


Fig. I.7 Jaguar CTS en la capa intermedia

El EAS y el servidor de base de datos pueden estar instalados en la misma máquina o en máquinas distintas. Además, el EAS proporciona servicios extras y conexiones a diferentes servidores.

Jaguar CTS proporciona un marco de trabajo para desarrollar la lógica de la capa intermedia de aplicaciones distribuidas basadas en componentes y simplifica la creación y administración de aplicaciones de Internet, que dan servicio a miles de clientes simultáneamente.

También proporciona una administración adecuada de sesiones de clientes, seguridad, hilos, conexiones a bases de datos en la tercer capa, así como el flujo de las transacciones, sin necesidad de que el desarrollador de componentes tenga un conocimiento especializado.

La escalabilidad de Jaguar, así como su independencia de plataformas permiten desarrollar aplicaciones en un sólo y económico procesador y después desplegar la aplicación en un servidor multi-procesador de escala empresarial.

Jaguar sirve para desarrollar aplicaciones con un gran volumen de transacciones dentro de Internet. Estas aplicaciones implican actualizaciones dinámicas en una sola dirección y en dos direcciones en tiempo real para información crítica.

También se pueden migrar las aplicaciones tradicionales cliente/servidor a aplicaciones Jaguar multi-capas.

A continuación se presentan las principales características de Jaguar CTS:

- Mecanismo de ejecución escalable, multi-hilos y de plataforma independiente.
- Soporte de envio para los modelos de componentes más importantes, como JavaBeans, PowerBuilder, Java, ActiveX y C/C++
- Soporte HTML dinámico utilizando servlets de Java y PowerDynamo.
- Estrecha integración con los ambientes de desarrollo de PowerBuilder y PowerJ.
- Sesiones de cliente transparentes y administración de los ciclos de vida de los componentes.
- Cachés de conexión que permiten reutilizar las conexiones a bases de datos remotas.
- Servicio de resolución de nombres de componentes.
- Manejo de transacciones para simplificar el diseño e implementación de las aplicaciones.
- Características transparentes de seguridad en hilos para simplificar el uso compartido de datos y recursos.
- Soporte de conjuntos de resultados para permitir una recuperación eficiente de datos tabulares en las aplicaciones cliente.
- Seguridad declarativa basada en perfiles para restringir las conexiones del cliente y de los componentes.
- Seguridad basada en identidad para restringir llamadas entre componentes.
- Ambiente de ejecución para componentes o servlets.

- Administra las peticiones del cliente a través de los protocolos:
  - IIOP : Protocolo inter ORB de Internet (default puerto 9000)
  - TDS : Servicio de datos tabulares (default puerto 7878)
  - HTTP: Lenguaje de hipertexto (default puerto 8080)
- Crea instancias de los componentes y los servlets de Java.
- Los servidores que se encuentran en la misma máquina host comparten el mismo almacén de datos.
- Utiliza Jaguar Manager, que es una herramienta gráfica de administración que:
  - Maneja el ciclo de vida de los componentes: define de que componentes se han creado instancias, cuales han sido limitados a los clientes o destruidos, y soporta poleo de instancias.
  - Administra las transacciones: permite definir una semántica transaccional para el componente.
  - Administra los hilos: los componentes que comparten recursos deben ser codificados o configurados para impedir un estado inconsistente.

En los siguientes capítulos se verá con mayor profundidad el funcionamiento de Jaguar y de sus componentes.

"Divide las dificultades que examines,
en tantas partes como sea posible
para hallar la mejor solución."

René Descartes.

Capítulo II Diseño de componentes

Como se dijo en el capítulo anterior, el servidor Jaguar se encuentra dentro del EAServer, en el cual es recomendable que los componentes instalados estén construidos sólo con Java o C++ y que la aplicación esté diseñada en base a una arquitectura de n-capas.

Los puntos importantes en el uso del *EAS* son el análisis, el desarrollo y la implementación utilizando los conceptos de la metodología orientada a objetos. Por lo cual se revisarán estos conceptos.

II.1 Programación orientada a objetos

Es un método de implementación en el cual los programas se organizan como un conjunto de objetos asociados entre sí, y cada uno representa una instancia que pertenece a una jerarquía de clases unidas por una relación de herencia. Este tipo de programación permite separar los diferentes componentes de un programa, simplificando su elaboración, depuración y mejoramiento. Cada parte del programa es tratado como un objeto independiente que integra a los procedimientos, las variables y los datos referentes a éste, y que además tiene ciertas características.

Los objetos se componen de tres partes fundamentales:

- **Métodos:** son funciones que permiten que el objeto realice cierta actividad y proporcione un servicio durante la ejecución de la aplicación. También determinan la forma en que el objeto responde al recibir un mensaje.

- **Eventos:** son acciones mediante las cuales el objeto reconoce que se está interactuando con él. De esta forma el objeto se activa y responde a un evento según lo programado en su código.

- **Propiedades:** son las características del objeto, en caso de que sea visible. Sus propiedades son su aspecto: color, tamaño, posición, etc.

Los objetos se comunican para trabajar en conjunto. Cuando un objeto se quiere comunicar con otro, le envía un **mensaje** con los datos. El objeto emisor no necesita saber la forma en que el objeto receptor realizar la acción. El conjunto de mensajes a los que un objeto puede responder, es el protocolo del objeto.

Se llama **instancia** a todo objeto que se deriva de otro. De esta forma, todos los

objetos son instancias de algún otro, menos la clase *Object* que es la madre de todas.

La **clase** es la descripción del objeto. Está formada de varios métodos y datos que forman las características del objeto. La definición de clases permite trabajar con código reutilizable, ya que a partir de una clase se puede crear una instancia y reutilizar el código escrito para ésta, sin tener que volverlo a escribir para la instancia, puesto que ésta toma el patrón de la clase padre.

Las principales características de la programación orientada a objetos son :

- **Herencia:** es un mecanismo para compartir automáticamente métodos y datos entre las clases, las subclases y los objetos. Permite crear nuevas clases con variaciones respecto a su clase padre. Existen dos clases de herencias:
  - *Herencia simple:* una subclase puede heredar datos y métodos de una clase simple y añadir o quitar ciertos comportamientos.
  - *Herencia múltiple:* es la posibilidad de tomar métodos y datos de varias clases simultáneamente.
- **Encapsulación:** define el comportamiento de una clase o de un objeto que tiene varios tipos de métodos y de datos. Sólo se puede acceder al objeto mediante mensajes, y a los datos a través de los métodos del objeto o clase.
- **Polimorfismo:** los objetos responden a los mensajes enviados. Un mismo mensaje puede generar distintas acciones según el objeto destinatario, mientras que el objeto emisor se desentiende de los detalles de la ejecución.

II.2 Definición de componente

Los componentes son módulos reutilizables de código que ejecutan tareas o métodos relacionados dentro de una interfaz definida previamente. Los componentes Jaguar se instalan en el servidor EAS. Sus funciones son contener los métodos que realizan la lógica del negocio y acceder a las fuentes de datos. Pueden ser distribuidos en una red, incluyendo Internet e Intranet, en diferentes servidores, y una vez instalados, varias aplicaciones independientes los pueden utilizar. Debido a que los componentes se localizan en el servidor, no pueden contener métodos para desplegar gráficos o interfases de usuario. Esto significa, que son heredados sólo de objetos no visuales.

En el Jaguar Manager, los componentes se pueden definir y editar, además de que se puede navegar gráficamente en sus interfases, las cuales definen los métodos que el cliente puede invocar. A pesar de que Jaguar almacena la información de la interfaz en CORBA IDL, no se necesita saber IDL para definir interfases desde el Jaguar Manager. Se pueden definir usando varias técnicas, como las que se describen a continuación:

- **Importación de interfases desde archivos de Java compilados:** en *Jaguar Manager se puede crear la definición de un componente leyendo las* definiciones del método desde una clase de Java compilada o desde un archivo de interfaz. El proceso de importación crea la interfaz IDL correspondiente en el almacén de interfases de Jaguar. Esto sirve para poder ejecutar las clases de Java existentes como componentes Java dentro de Jaguar. También se puede importar una interfaz Java para definir un componente de cualquier tipo.

- **Importación de interfases desde componentes ActiveX registrados:** Jaguar Manager puede importar firmas de métodos de ActiveX o de archivos de librerías de tipos, lo cual, por ejemplo, permite adaptar un servidor ActiveX no visual como un componente Jaguar.

- **Definición de módulos e interfases en IDL:** Jaguar almacena todos los componentes en módulos de Lenguaje de Definición de Interfase (IDL). En el Jaguar Manager, el folder de interfases muestra todos los módulos disponibles en el almacén de interfases en Jaguar.

II.2.1 Arquitecturas soportadas por Jaguar CTS

En la siguiente figura se muestran varios tipos de clientes: Java, HTML, PowerBuilder, COM, CORBA; los cuales se comunican por medio del protocolo común IIOP desde una red determinada, con el servidor Jaguar. La interacción entre diferentes tipos de componentes dentro del *EAS* se lleva a cabo gracias a la arquitectura de componentes distribuidos *CORBA*.

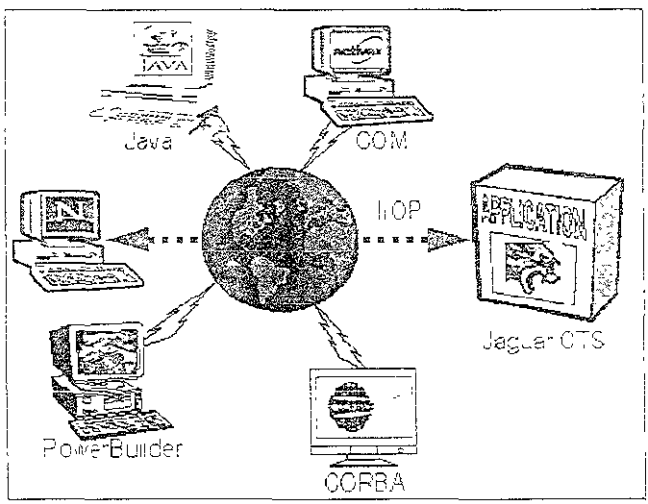


Fig. II.1 Arquitectura de Jaguar

La siguiente figura muestra un caso en el que interactúan componentes distintos a los de Jaguar y ORBs de otros proveedores, con el servidor Jaguar. Esta característica del EAS permite que las aplicaciones creadas por terceros utilicen el servidor Jaguar.

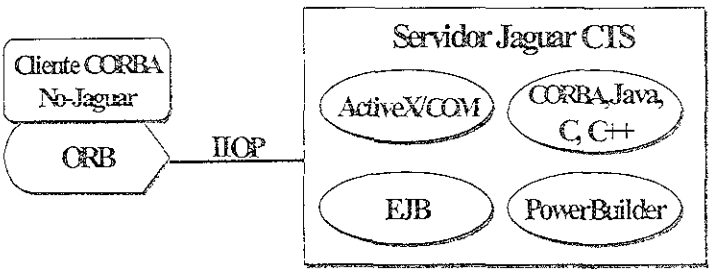


Fig. II.2 Clientes no-Jaguar accediendo a componentes Jaguar

También permite desarrollar componentes con lógica encapsulada, que se podrían utilizar en cualquier aplicación. Por ésto, es recomendable planear y desarrollar los sistemas con una metodología orientada a objetos. El encapsulamiento de los componentes les permite poderse cambiar a cualquier plataforma y tener un uso flexible.

La comunicación bidirecciones entre Jaguar y otros *ORB*s y/o componentes, hace de los componentes y del servidor un medio de enlace entre diferentes plataformas y tecnologías. Por lo tanto, cualquier componente Jaguar puede dar y solicitar servicios a componentes externos.

Además, un componente de cualquier modelo puede ejecutar componentes de otro, utilizando llamadas entre componentes sin necesidad de un software de pasarela adicional. Como se ha visto, Jaguar soporta componentes Java, PowerBuiler, C++, ActiveX y C. Si se desarrollan componentes con PowerBuilder o PowerJ, se pueden crear y desplegar componentes directamente desde ese ambiente de desarrollo.

En el Jaguar Manager, los paquetes permiten agrupar componentes relacionados como unidades lógicas. Cuando se desarrolla con otras herramientas, se deben definir los paquetes y los componentes en el Jaguar Manager. Jaguar también proporciona una interfaz de objetos compartidos que permite a los componentes almacenar referencias a datos compartidos.

II.2.2 Soporte en el cliente de stubs y proxies

Un stub es una clase de Java generada por Jaguar Manager, que actúa como intermediario entre un componente que se encuentra en el servidor Jaguar, y el cliente que lo invoca. Compilado y ligado con una aplicación cliente en Java, el stub se comunica con Jaguar para instanciar y llamar a un método del componente. Los stubs hacen que los componentes remotos parezcan locales.

El soporte de componentes Jaguar del lado del servidor y de stubs del lado del cliente son independientes. Un cliente Jaguar puede ejecutar cualquier tipo de componente. Internamente, el objeto stub o proxy establece una conexión entre el servidor Jaguar y el cliente remoto. Todos los modelos stub/proxy requieren parámetros de autentificación de usuarios para poder instanciar un objeto stub o proxy. El protocolo de comunicación también se determina por el tiempo que el objeto es instanciado. Una vez que ya existe, se ocultan todos los detalles de la comunicación en la red.

El nombre del usuario determina a que componentes puede acceder la sesión del cliente. Si un componente no permite el acceso al usuario, entonces intenta generar una falla del stub. Los stubs o proxys utilizan el Protocolo de Invocación Inter-ORB (IIOP) para comunicarse con el servidor Jaguar, mientras que los clientes MASP utilizan el protocolo de flujo de datos tabulares (TDS) de Sybase.

Los stubs y proxies están disponibles para :

- **Java (CORBA y EJB):** cualquier componente se puede invocar por medio de una clase stub en Java. Jaguar Manager genera código fuente para los stubs en Java. En el tiempo de ejecución, el programa cliente crea instancias del stub. Cuando se llama a los métodos en la clase stub, éste invoca transparentemente al método del componente en el servidor Jaguar. Usando páginas HTML, applets de Java y el soporte de Jaguar para construir páginas HTTP, se pueden crear aplicaciones que no requieren de instalación en la máquina cliente, y que sólo necesiten de un navegador Web que soporte Java. Jaguar soporta tres modelos de cliente Java:

  - **Enterprise Java Beans:** se usan las clases JavaSoft EJB 1.0 (**javax.ejb**) y los stubs de Jaguar EJB 1.0 para llamar a los métodos de los componentes Jaguar.

  - **Java-CORBA:** se usa un ORB de Java compatible con CORBA de Jaguar o cualquier otro ORB de Java compatible con CORBA para crear instancias de stubs. Las firmas de los métodos del stub se direccionan desde la definición de la interfaz de los componentes basados en la especificación CORBA de las ligas del lenguaje IDL de Java.

  - **Jaguar 1.1** se usa la interfaz del cliente Java en la versión Jaguar 1.1., la cual es obsoleta y sólo es proporcionada por compatibilidad con clientes Java desarrollados para Jaguar 1.1.

- **PowerBuilder:** PowerBuilder 7.0 o posterior permite generar objetos no visuales que actúan como proxys para componentes Jaguar. Al usar un proxy, se pueden llamar métodos de componentes como si fueran métodos de objetos no visuales locales.

- **C++ (CORBA):** los programas usan el ORB de C++ compatible con CORBA de Jaguar o cualquier otro ORB de C++ compatible con CORBA para instanciar los stubs. Las firmas de los métodos stub se mapean desde la definición de la interfaz de los componentes, basados en la especificación CORBA de las ligas del lenguaje IDL-de C++.

- **ActiveX:** los programas invocan a los componentes Jaguar usando proxys ActiveX de Jaguar, los cuales permiten llamarlos desde herramientas como Visual Basic. En el servidor, el proxy ActiveX permite invocar cualquier componente Jaguar desde un componente ActiveX. Jaguar Manager genera la información de la librería de tipos que es requerida para registrar la interfaz del componente con la herramienta de desarrollo. El proxy ActiveX usa el ORB del cliente C++ de Jaguar para comunicarse con los servidores Jaguar.

- **Métodos como Stored Procedures (MASP):** Jaguar proporciona una interfaz para ejecutar los métodos de los componentes como si fueran procedimientos almacenados en una base de datos de un servidor de Sybase. Esta interfaz se puede usar para llamar a componentes Jaguar desde una herramienta de construcción de interfaz de usuario y desde código para bases de datos que no soportan ActiveX.

II.2.3 Soporte de componentes del lado del Servidor

El esqueleto (skeleton) es una clase que se encuentra en el servidor, la cual sirve para que un ORB haga peticiones a un objeto del cual no conoce su implementación al tiempo de compilación. Actúa como la interfaz entre Jaguar y el código que implementa el método. Los esqueletos son compilados y ligados con cada componente, y en el tiempo de ejecución le permiten a Jaguar localizar e invocar el método apropiado. Los esqueletos siempre están del lado del servidor.

Jaguar da soporte a los principales modelos de componentes, incluyendo a:

- **Enterprise JavaBeans (EJB) :** Jaguar soporta componentes Java que siguen la especificación EJB versión 1.0. Un EJB es un componentes no visual y transaccional que es implementado en Java.

- **Java-CORBA:** estos componentes siguen el modelo CORBA y usan las interfases estándares CORBA para el manejo transaccional. La mayoría de las clases Java con comportamiento no visual se pueden adaptar para que funcionen como componentes de Jaguar.

- **Componentes PowerBuilder no visuales:** con PowerBuilder 7.0 o posterior, se pueden crear objetos no visuales que se ejecutan en un servidor Jaguar como componentes Jaguar. También se pueden crear proxys para componentes Jaguar y usar los proxys en las aplicaciones cliente de PowerBuilder.

- ***Componentes CORBA C++:*** *son clases C++ que contienen métodos con* prototipos similares a la interfaz de los componentes Jaguar.

- **ActiveX:** cualquier componente ActiveX no visual se puede instalar como un componente Jaguar. Jaguar usa soporte COM y la automatización ActiveX para ejecutar los métodos de los componentes ActiveX. En consecuencia, todos los componentes ActiveX de Jaguar deben soportar la interfaz de automatización COM. Varias herramientas de desarrollo de aplicaciones, como Visual Basic, se pueden usar para crear componentes ActiveX compatibles con Jaguar. Jaguar implementa subconjuntos de la interfaz ActiveX del Servidor de Transacciones de Microsoft (MTS) con el objetivo de proporcionar una compatibilidad completa.

- **Componentes C:** Jaguar proporciona un modelo de pseudo componentes en C que se puede usar para adaptar los procedimientos en C existentes como componentes Jaguar. Los componentes C son librerías dinámicas (DLLs) o librerías compartidas de UNIX que contienen métodos C o C++ y métodos del esqueleto que contienen funciones de C que reciben los parámetros de llamada de procedimiento remota (RPC) e invocan a métodos de componentes C. Las clases C++ pueden funcionar como componentes C++.

II.2.4 Manejo del ciclo de vida del componente

El ciclo de vida de un componente determina la forma en la que se crean sus instancias, en que se ligan a sesiones cliente, y finalmente se destruyen. En los casos más simples, se crea una instancia por cada stub, y dicha instancia es destruida cuando el cliente lo solicita explícitamente o cuando se desconecta.

Jaguar maneja los ciclos de vida de las sesiones cliente y de los componentes sin que el desarrollador necesite tener un conocimiento especializado.

El ciclo de vida de un componente está diseñado para maximizar sus propiedades para reutilizar los recursos del servidor y minimizar la posibilidad de que una aplicación del cliente monopolice los recursos del servidor. Para lograrlo, Jaguar cuenta con el poleo de instancias de componentes y la desactivación temprana, que se explican a continuación:

- **Poleo de instancias:** permite que una sola instancia del componente sirva para varios clientes. El ciclo de vida del componente contiene la activación y desactivación de una instancia. La primera une una instancia a un cliente y la segunda indica que la instancia se ha desligado. El poleo de instancias elimina el desperdicio de recursos que tiene lugar por la creación constante de instancias de

componentes.

- **Desactivación temprana:** permite que un método del componente especifique el momento en que ocurre la desactivación, evita que la aplicación del cliente mantenga reservados los recursos asociados a una instancia del componente, y permite que ésta sirva a más clientes dentro de un tiempo establecido. Para obtener la desactivación temprana, se puede codificar o configurar el componente.

Un componente que es desactivado después de cada llamada al método y que soporta el poleo de instancias, es conocido como **stateless** (sin estado), porque el estado del componente se vuelve a configurar entre cada cambio de transacción y activación.

La desactivación temprana y el poleo de instancias generan una mayor escalabilidad, al permitir que más clientes usen un número estático de instancias, por lo que un diseño basado en componentes stateless ofrece mayor escalabilidad. También le permite a Jaguar mantener un caché de instancias de componentes y ligarlas a una sesión del cliente, según sea necesario. El poleo de instancias requiere de las siguientes características en el componente:

- El componente debe proporcionar los métodos *activate* y *deactivate*. Jaguar llama al método *activate* un poco antes de que una instancia sea ligada a una sesión del cliente.
- El método *activate* debe poder reconfigurar el componente de acuerdo a un estado establecido. Jaguar llama al método *deactivate* un poco antes de que una instancia sea desconectada de la sesión del cliente. Es decir, antes de quedar nuevamente desocupada.

Para los componentes que soportan transacciones Jaguar, el tiempo entre las llamadas de Jaguar a los métodos *activate* y *deactivate* coincide con el inicio y el final de la participación de esa instancia en la transacción Jaguar.

Para componentes Java y AciveX, se puede implementar una interfaz de control del ciclo de vida que verifique si existe poleo de instancias de los componentes. Esta interfaz también proporciona métodos *activate* y *deactivate,* que son llamados para indicar transiciones de estado en el tiempo de vida de la instancia del componente.

Para soportar el poleo de instancias, el código que responde a los eventos de activación debe restaurar el estado inicial del componente, como si estuviera recién creado. Tanto las interfases Java como las ActiveX tienen métodos que permiten a una instancia rechazar selectivamente el poleo: *canReuse* en Java y *canBePooled* en ActiveX.

II.2.5 Balanceo de cargas y soporte de fallas

En el caso más sencillo, los componentes de aplicaciones se despliegan a un solo servidor y todos los clientes se conectan a ese servidor para ejecutar la lógica del

negocio del componente. Este caso funciona bien mientras el número de clientes que acceden simultáneamente no aumente.
Para aplicaciones con miles de clientes, se puede definir un grupo (cluster) Jaguar con varios servidores que ejecuten los componentes de las aplicaciones. El grupo posibilita el balanceo de cargas y el soporte de fallas de la siguiente manera:

- **Balanceo de cargas:** optimiza el funcionamiento del grupo Jaguar al ajustar las cargas entre los servidores, basado en la medición de las cargas y las reglas de distribución, o en un algoritmo al azar. Si el cliente resuelve el nombre del componente, entonces el servidor de nombres regresa varias direcciones de servidores candidatos. Cuando es el algoritmo el que distribuye al azar las cargas de procesamiento en los servidores del grupo, el ORB del cliente prueba las direcciones en orden aleatorio. Cuando se especifican las medidas de las cargas y las reglas de distribución, las cargas son distribuidas de acuerdo a las cargas actuales de cada servidor.

- **Soporte de fallas:** los componentes pueden configurarse para permitir un soporte de fallas automático y transparente para las aplicaciones del cliente. Si un componente permite un soporte de fallas automático, entonces el ORB del cliente se reconecta automáticamente a otro servidor del grupo cuando alguno queda fuera de línea.

II.3 Tipos de componentes Jaguar

En Jaguar existen tres tipos de componentes. Cada tipo está diseñado para un propósito diferente y cada uno tiene sus puntos fuertes y débiles. La implementación y creación de instancias de un componente en Jaguar, están directamente relacionadas con las propiedades asociadas al componente. A continuación se revisan los tres tipos.

II.3.1 Componente estándar

Es el tipo de componente más común. Generalmente contiene la implementación de la lógica del negocio y participa en transacciones a bases de datos controladas por Jaguar. Dependiendo de las propiedades de los componentes, Jaguar crea varias instancias de un componente estándar para responder a las solicitudes simultáneas hechas al componente.

II.3.2 Componente compartido

Es una instancia activa del componente. Debido a que sólo hay una, la memoria no está realmente compartida, sino que el componente es accesible a las llamadas de los servidores internos y externos. No significa que se creen varias instancias del componente, que compartan el mismo espacio de memoria, sino que se crea sólo una instancia del componente. Por ejemplo, se puede usar un componente compartido

para almacenar una lista estática de datos, aunque éstos provengan de una base de datos. Jaguar crea una sola instancia de un componente compartido, sin considerar el número de clientes o componentes que lo soliciten. Uno de sus usos, es establecer un caché que proporcione acceso a los datos que son solicitados constantemente. Esta técnica evita consultar repetidamente la base de datos, la cual puede ser inicialmente llenada y periódicamente actualizada a través de un componente de servicio.

II.3.3 Componente de Servicio

Es otra clase de componentes especiales, los cuales empiezan a funcionar cuando el servidor inicia y terminan de ejecutarse cuando el servidor finaliza. Son diseñados para ejecutarse en la parte no visual de la aplicación (background) y son controlados por el servidor Jaguar. Realizan un proceso en lotes (batch) que no es invocado por la aplicación cliente, sino cuando el servidor inicia. Implementan tres métodos: start(), run() y stop(), los cuales afectan al ciclo de vida del componente.

- **Método start:** Se invoca cuando el servidor de Jaguar se inicia o cuando es reiniciado físicamente. Refrescar el componente de servicio o el servidor de Jaguar desde el Jaguar Manager no hace que el método start() se ejecute otra vez. Inmediatamente después de regresar del método start(), el servidor invoca al método run().

- **Método run:** Es en donde se realiza la mayoría del proceso. Una vez que el método run() ha terminado de ejecutarse, el componente de servicio es finalizado y no realiza otro proceso a menos que el servidor de Jaguar sea reiniciado. El desarrollador es responsable de escribir el código de este método, que hace que el componente sea accesible para cualquier cliente o servidor basado en componentes.

- **Método stop:** se invoca cuando el servidor es actualizado o apagado, o el componente de servicio se actualiza a sí mismo.

Este tipo de componentes proporcionan servicios comunes para clientes de Jaguar, así como para otros componentesr. Por ejemplo, se pueden crear componentes de servicio para realizar las siguientes tareas :

- Mantener copias del caché para tablas de bases de datos usadas frecuentemente.

- Mover o replicar datos entre varias fuentes durante el tiempo en el que el servidor está desocupado.

- Administrar archivos de registro (log) de aplicaciones específicas.

Los componentes de servicio son como cualquier componente Jaguar, excepto que:

- Deben implementar la interfaz *CtsServices::GenericService*, que contiene a los métodos start(), run() y stop().

- Las instancias se cargan y se inicializan cuando lo hace el servidor Jaguar.

- Se pueden ejecutar independientemente de la interacción con el cliente.

El componente de servicio puede implementar interfases adicionales. Los clientes Jaguar, los servlets y otros componentes pueden ejecutar los métodos de los componentes de servicio como si fueran los de cualquier otro componente, con sólo una excepción: los clientes no pueden invocar métodos hasta que el método start() finalice. Esta restricción permite hacer la inicialización requerida en start() sin preocuparse por la sincronización de hilos.

Con este tipo de componentes se pueden efectuar tareas calendarizadas para desactivar al componente durante cierto tiempo. Por ejemplo, si se mantiene una lista semi-estática de datos, quizás se quieran usar componentes de servicio que son accesibles exteriormente, pero mientras no se usen, estarán inactivos en el servidor. En cualquier momento (el periodo de tiempo es configurable), el componente de servicio se activa a sí mismo y actualiza su información desde el servidor de bases de datos.

Los componentes de servicio y compartidos se pueden combinar, de forma que se tenga un componente de servicio compartido que es instanciado una sola vez y automáticamente cuando el servidor inicia y que se desactiva cuando no está en uso.

II.3.4 Propiedades de los componentes en Jaguar Manager

Dentro del Jaguar Manager se pueden ver y modificar las propiedades de un componente mediante una caja de diálogo que aparece al marcar el componente con el botón derecho del ratón y escogiendo la opción de Propiedades, como se observa en la siguiente ilustración.

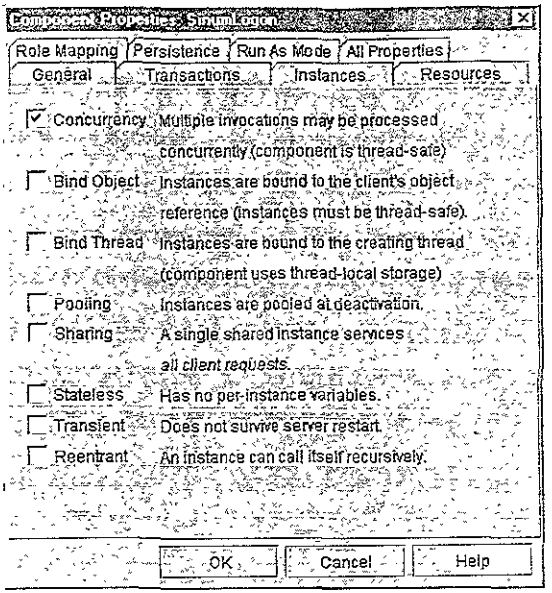


Fig. II.3 Propiedades de los componentes en la pestaña de instancias

Las propiedades de las instancias son importantes para el proceso de creación de instancias y de acceso a los componentes. Cada propiedad se explica a continuación y se describen los aspectos individuales, la manera en que cada una se relaciona con las otras, y el efecto de red resultante de sus combinaciones.

- **Recursivo (Reentrant):** se aplica sólo si el componente es un EJB. Las especificaciones del EJB prohíben explícitamente el uso de código recursivo.

- **Temporal (Transient):** es otra propiedad específica de componentes EJB. Indica que el objeto es válido sólo durante la sesión del cliente y define que tipo de objeto es el componente (entidad, sesión formal o sesión informal).

- **Stateless (Sin estado):** si se activa, Jaguar no dispara los eventos activate() o deactivate() del componente con el fin de mejorar su funcionamiento. Marcar esta propiedad, no permite la demarcación y desactivación automáticas, ni necesariamente desactiva la instanciación del componente después de una llamada al método.

- **Poleo (Pooling):** al activarse, Jaguar pospone indefinidamente la destrucción física de la instancia del componente. El poleo mejora el funcionamiento de los componentes, ya que Jaguar no pierde tiempo creando instancias y

destruyendo recursos después de que cada cliente ha terminado de trabajar con un componente. Además Jaguar no invoca a la función *canReuse()* o al evento *canBePooled()*. Este evento sólo se dispara cuando el poleo está desactivado, dando la oportunidad al pool de controlar el acceso del componente al pool.

- **Objeto enlazado (Bind Object):** es una característica única del EAServer. Permite que varios clientes llamen a una misma instancia del componente a través de diferentes servidores Jaguar, a diferencia de un objeto compartido que está ligado a un solo servidor. La idea es que el componente almacene los datos que son solicitados constantemente en los servidores del grupo, y que los servicios que implican transacciones aseguren que los datos serán replicados para cada instancia en todos los servidores.

- **Hilo enlazado (Bind Thread):** al activarla, se indica que cualquier llamada a un método del componente debe usar el mismo hilo que usa Jaguar para crear la instancia del componente, ya que los datos son guardados en ese hilo mediante el almacenamiento local de hilos (TLS). Esta propiedad es crítica para componentes Power Buider y COM construidos para funcionar en Windows NT. Si el componente se encuentra en cualquier ambiente UNIX, esta propiedad es ignorada, a pesar de que el componente haya sido desarrollado con PowerBuilder.

- **Compartir y Concurrencia (Shering, Concurrency):** estas propiedades son más fáciles de explicar de manera conjunta, debido a su interdependencia y a la forma de implementar sus distintas combinaciones. "Compartir" establece la cantidad de instancias de un componente que Jaguar puede crear. Cuando es activada, Jaguar puede crear una instancia única de la clase del componente. Mientras que "Concurrencia" determina si varios clientes pueden invocar simultáneamente a un método de la misma instancia. Cuando se activa, Jaguar permite que los métodos de una misma instancia operen en hilos separados, y por lo tanto, que sean accedidos por más de un cliente al mismo tiempo. EJB restringe explícitamente a que una sola instancia atienda simultáneamente a las peticiones de varios clientes.

A continuación se presentan algunos ejemplos del uso en conjunto de las propiedades "Concurrencia" y "Compartir". Para cada escenario se asume que hay dos clientes tratando de interactuar simultáneamente con la misma clase del componente.

- **Concurrencia: desactivada / Compartir: desactivada**

Jaguar crea varias instancias del componente para varios clientes, pero sólo una podrá estar activa en cualquier momento y no se podrán crear componentes adicionales si alguna está activa, puesto que "Concurrencia" está desactivada.

Por ejemplo, el método A() del Cliente 1 será ejecutado en la Instancia 1, mientras que el método A() del Cliente 2 será ejecutado en la Instancia 2. Pero el Cliente 2, de cualquier forma, tendrá que esperar a que la llamada al método de la Instancia 1 termine, para empezar la llamada al método de la Instancia 2 .

- **Concurrencia: activada / Compartir: desactivada**

Si "Concurrencia" está activada, "Compartir" desactivada y ambos clientes están tratando de ejecutar métodos al mismo tiempo en la clase del componente, Jaguar creará varias instancias del componente y ejecutará cada método en paralelo o concurrentemente en hilos separados en instancias separadas.

- **Concurrencia: activada / Compartir: activada**

Jaguar crea una sola instancia de la clase del componente que ambos clientes ejecutarán, y en la cual varios clientes podrán trabajar en cualquier momento. PowerBuider no soporta esta combinación, puesto que un componente PowerBuilder no se encuentra seguro de hilo *(thread-safe).*

- **Concurrencia: desactivada / Compartir: activada**

Jaguar crea una sola instancia que ambos clientes ejecutarán. Sin embargo, sólo un cliente podrá estar trabajando activamente dentro de un componente. Todas las demás solicitudes son formadas en una cola y ejecutadas en forma serial.

Los hilos enlazados juegan un papel importante en conjunto con estas dos propiedades. Si "Compartir" e "Hilos enlazados" están activadas, se obliga a Jaguar a invalidar la opción de "Concurrencia" sin importar que se encuentre activada, puesto que se obliga al componente a permanecer asociado al hilo que lo creó. Con "Compartir" activada (máximo se tiene una instancia), es ilógico e imposible tener varios hilos ejecutándose en paralelo con el componente.

II.3.5 Componentes *stateful* y *stateless*

Stateful y *Stateless* son términos que se refieren al tiempo de vida o a la duración de un componente después de haber invocado al método desde una conexión del cliente y de que la ejecución del método ha finalizado.

Un componente que puede permanecer activo entre invocaciones consecutivas a uno de sus métodos, es llamado *stateful.* Un componente que es desactivado después de cada llamada al método es conocido como *stateless.*

II.3.5.1 Componentes stateful

Los componentes *stateful* pueden usar variables de instancia que almacenan los datos que son constantemente accedidos por el cliente entre las invocaciones a los métodos. Estos componentes están dedicados a una sola sesión del cliente.

Debido a que la desactivación se hace a voluntad de la aplicación cliente, se puede configurar la propiedad "tiempo fuera" de la instancia de los componentes *stateful*, de manera que un cliente no la monopolice indefinidamente.

Los componentes *stateful* pueden colocarse en el *pool* después de haberse desactivado, de modo que una instancia sirva para varias sesiones del cliente. Durante las llamadas al método desde el cliente, el servidor está utilizando recursos (memoria) para mantener el estado de los componentes, por lo cual este enfoque no escala correctamente cuando aumenta el número de usuarios.

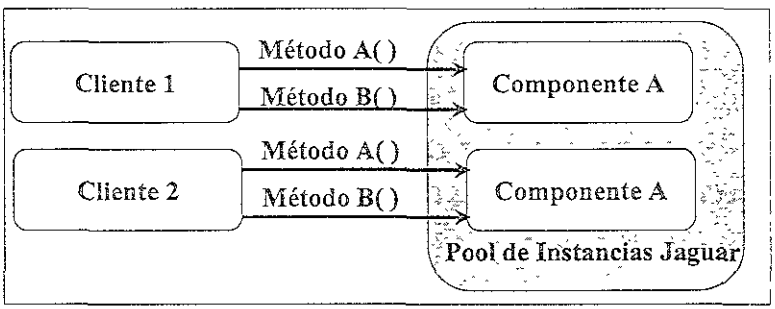


Fig. II.4 Clientes invocando métodos en un ambiente stateful

En la figura anterior se tienen dos clientes conectándose a una instancia del componente "A". Cada cliente invoca al método A(). El cliente espera a que el usuario ingrese datos antes de invocar al método B(). Ambos métodos se ejecutan en la misma instancia y el componente permanece dedicado a la sesión del cliente que lo invocó. En el método A() se puede almacenar el estado que se utilizará en las llamadas al método B(). De cualquier forma, estas instancias quedan ligadas a la sesión del cliente sin importar cuanto tiempo le tome al usuario responder, y siempre y cuando no exceda el tiempo establecido en la propiedad de tiempo fuera del componente (*timeout*=0 significa: indefinidamente).

Los diseños *stateful* requieren que el servidor aparte los recursos necesarios para un componente aunque el cliente asociado no esté involucrado activamente con éste. Es debido a este requerimiento, que un diseño *stateful* no escala bien cuando se incrementa el número de conexiones. Sin embargo, la capacidad de un componente

stateful de conservar los datos de la instancia entre las llamadas a los métodos, representa una ventaja para las reglas del negocio que así lo requieran.

II.3.5.2 Componentes stateless

Para que un componente sea *stateless*, debe cumplir estos dos puntos:

- El componente está configurado o implementado para ser desactivado después de cada invocación a un método. En el Jaguar Manager se puede habilitar la propiedad "Desactivación/Demarcación Automáticas" para el componente. Alternativamente, se puede implementar el componente de forma que llame al evento **completeWork** o a **rollbackWork** en cada método.

- La opción "Pooling" está activada en la ventana de Propiedades del componente.

Los componentes stateless no pueden utilizar los datos de una instancia específica para acumular datos entre invocaciones a un método. Sin embargo, en algunas situaciones ésto si es necesario. Por ejemplo: un componente llamado **OrdenCompra** podría tener un método agregaArticulo() que es llamado repetidamente para especificar el contenido de una orden.

En lugar de usar los datos de una instancia específica, se pueden usar cualquiera de las siguientes alternativas para acumular los datos :

- **Acumular datos en una base de datos remota:** la técnica más utilizada es tener un caché de conexiones y guardar los datos en un base de datos remota. Si se tiene un componente en un grupo, éste podría operar en varios servidores y la base de datos sería un almacén central.

- **Acumular datos en el cliente:** se crea una estructura de datos que se pasa en cada invocación al método y que contiene los datos acumulados. Esta técnica sólo resulta práctica con pocos datos. El envío de muchos datos a través de la red, perjudicaría su funcionamiento.

- **Acumular datos en un archivo:** si los datos acumulados son pocos y con estructuras de datos simples, se pueden almacenar en un archivo local.

Un componente *stateless* no puede usar variables de instancia para mantener datos específicos del cliente entre invocaciones al método, lo cual no significa que no puedan usar variables de instancia.

Almacenar la referencia de un componente, es un ejemplo de la forma en que un componente *stateless* utiliza una variable de instancia, lo cual tiene las ventajas de que el esfuerzo de buscar al componente sólo ocurriría una vez y de que cada ejecución del método podría reutilizar la instancia. Otro ejemplo es la posibilidad que

tiene un componente de dividir la lógica redundante de los métodos privados o protegidos que son invocados desde un método público, manteniendo una firma pequeña de dichos métodos, puesto que éstos tienen acceso a las variables de instancia.

Si el estado de las variables de instancia necesita seguir igual entre las llamadas a los métodos del componente, antes de restablecerlas, sus valores deben almacenarse en otro lugar dentro del componente. De hecho, es muy frecuente que un componente *stateless* continúe con el mismo estado en cualquier lugar externo a Jaguar, como una base de datos o un archivo plano.

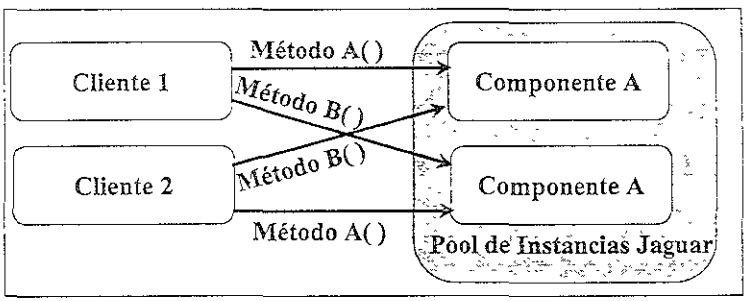


Fig. II.5 Clientes invocando métodos en un ambiente stateless

En la figura anterior se encuentran dos clientes invocando a los mismos métodos que en la figura II.4. El cliente sigue presentando al usuario el valor regresado del método A() y espera a que el usuario responda para invocar al segundo método; pero en esta ocasión, Jaguar no conserva ni la instancia del componente entre las llamadas, ni ningún valor del estado.

De hecho, si los componentes son multi-hilos y no son compartidos ("Concurrencia" = verdadero, "Compartir" = falso), el Cliente 1 y el Cliente 2 podrían estar, teóricamente, invocando al método A() y al método B() en diferentes instancias del componente al mismo tiempo, pero en distintos hilos. En esta situación, un diseño *stateless* permite que Jaguar maneje mejor las solicitudes que vayan llegando del cliente al método. Ahora Jaguar está en posición de determinar si es necesario crear otra instancia del componente para satisfacer las solicitudes de un nuevo cliente o si tomará algún hilo que esté disponible en el *pool* de hilos y lo asociará con el método solicitado, conservando los recursos del sistema.

Un componente *stateless* está ligado a un cliente sólo durante la ejecución de un

método público. Al concluir el método público, el componente se libera y generalmente se vuelve a colocar en el *pool* de instancias, a disposición del próximo cliente que solicite el componente.

Una de las razones por la que se pueden utilizar variables de instancia dentro de un componente *stateless*, es que éste sólo existe mientras dura la invocación al método público que lo originó. Sin embargo, un método público puede invocar a otros métodos públicos o protegidos a los que haga referencia y que utilicen variables de instancia de componentes.

A pesar de que durante la ejecución del evento *deactivate* no se solicite restablecer los datos anteriores de la variable de instancia, es recomendable inicializar las variables de instancia en el evento *activate* o limpiarlas en el evento *deactivate*. Si no se limpian sus valores, el siguiente cliente podría heredar el valor de las variables de instancia, lo cual podría acarrear errores.

II.3.6 Diferencias entre los componentes *stateful* y *stateless*

Normalmente un componente *stateful* tiene desactivada la propiedad Demarcación/Desactivación Automática, la propiedad de transacción activada, el valor de la propiedad de tiempo fuera de la instancia mayor que cero, y además debe tener un método para que los clientes puedan desactivar el componente. La siguiente figura muestra la pestaña "Transacciones" de las propiedades del componente en donde Demarcación/Desactivación Automática está seleccionada, lo cual convierte *stateless* al componente.

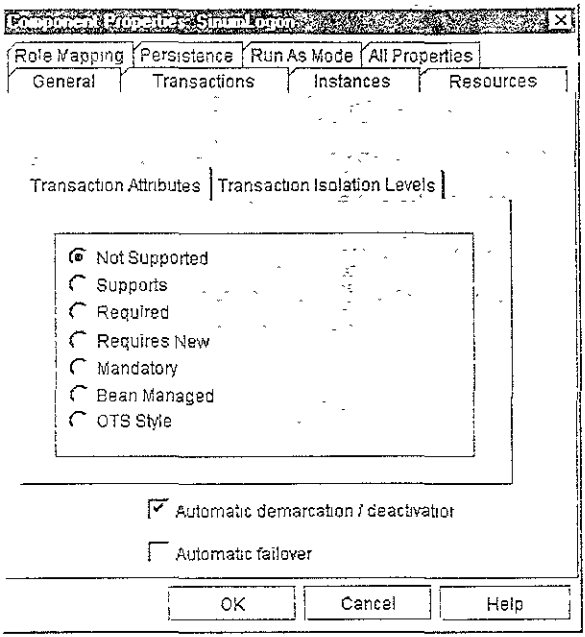


Fig. II.6 Pestaña de Propiedades de Transacciones del componente

En los componentes *stateful* es posible que la propiedad "Tiempo Fuera" tenga un valor de cero, que indica tiempo infinito, y significa que el componente nunca se desactivará a sí mismo, a menos que sea solicitado explícitamente por el cliente. En caso de que el cliente olvide desactivar el componente, la propiedad de tiempo fuera se ejecutará, siempre y cuando tenga un valor mayor que cero, y desactivará el componente automáticamente al accionar el evento *deactivate* durante el proceso. Si la propiedad de tiempo fuera no está activada, existe el peligro de que los clientes que hayan olvidado desactivar el componente con los métodos setComplete(), completeWork(), setAbort() o rollbackWork(), desperdicien los recursos del servidor, lo que provocará que el servidor Jaguar entre en un estado de inactividad por mucho tiempo.

Un componente *stateless* es más sencillo de definir que uno *stateful*. La única configuración requerida para crearlo es la activación de Demarcación/Desactivación Automática.

En la versión 3.5 del EAServer, existe una nueva propiedad del componente llamada com.sybase.jaguar.component.stateless. Al activarla se desactivan los eventos *activate* y *deactivate*. Sin embargo, utilizarla no provoca que el componente se desactive automáticamente después de llamar a un método.

II.3.7 Construcción de un componente *stateful*

Existen tres enfoques diferentes para construir un componente *stateful*. Todos tienen en común a la propiedad Demarcación/Desactivación Automática desactivada y difieren en la manera de interactuar con las transacciones de las bases de datos controladas por Jaguar.

El primer enfoque es utilizar el atributo de transacción *Not Supported* del componente *stateful*. En este escenario, el componente puede trabajar con una base de datos, inclusive puede actualizar datos, pero no puede participar en ninguna transacción de Jaguar. Es decir, que toda la lógica de la transacción es controlada por el desarrollador y Jaguar nunca ejecutará un comando *commit* o *rollback* en la base de datos. Es completamente la responsabilidad del desarrollador asegurarse de que el componente termine correctamente su transacción con la base de datos. Un componente que utiliza el atributo *Not Supported* no puede compartir su transacción con otro.

Otro enfoque es la *desactivación temprana*. Un componente *stateful* que la utiliza, configura el atributo de transacción de Jaguar como *Required* o como *Required New*. Por ejemplo, suponiendo que el componente tiene cuatro métodos setxxxxx() y un método llamado completeUpdate(boolean). El cliente invocará a cada uno de los métodos, en un orden lógico según la regla del negocio que se esté implementando. Después de que todos los datos hayan sido enviados al componente, el cliente invoca al método completeUpdate() para guardar los datos en la base de datos.

Después de haber ejecutado las sentencias SQL apropiadas, el componente llamará a setComplete()/completeWork() o a setAbort()/rollbackWork() según sea el caso, Jaguar elegirá llevar a cabo la transacción o deshacerla. Es importante notar que el componente será desactivado desde la sesión del cliente, perdiéndose el estado almacenado en memoria. Esta es la razón por la que el término "desactivación temprana" es utilizado, ya que Jaguar desactiva al componente, pero no al cliente.

El último enfoque para el diseño de componentes *stateful* es el híbrido. En éste, los métodos son invocados como lo habían sido anteriormente y cuando se llama al método completeUpdate(), éste llama a otro componente, generalmente *stateless*, para manejar el procesamiento de la transacción, pasando a todas las instancia los datos que el componente ha recolectado. En este caso, el manejo de la transacción es transferido al componente, que recibe la llamada del componente *stateful* de manera que éste último permanezca activo para la sesión del cliente.

II.3.8 Construcción de un componente *stateless*

Construir un componente *stateless* es sencillo, la parte complicada es mantener la información entre las llamadas al método. A continuación se describen diferentes técnicas de diseño para crear un componente *stateless* y manejar sus instancias de datos.

La siguiente declaración sirve como sinopsis de un problema ejemplo:

La información contenida en las variables de instancia creadas en un componente *stateless* durante una invocación al método, se pierde al concluir dicho método. Esta información podría necesitarse en futuras invocaciones al método.

Existen varias posibilidades para el diseño y administración de componentes *stateless*:

- Diseño del caché en el cliente.
- Diseño del caché en la base de datos.
- Diseño del caché en archivos planos.
- Diseño del caché en componentes compartidos.

A continuación se explican con más detalle cada una de estas opciones.

II.3.8.1 Diseño del caché en el cliente

Una conexión caché es una estructura interna de Jaguar que mantiene un poleo de las conexiones disponibles para un servidor que se encuentra en la tercer capa. Todas las conexiones en el caché deben compartir el mismo nombre de usuario y la misma clave de acceso, conectarse al mismo servidor de la tercer capa y usar la misma librería de conectividad.

Los clientes deben tener una variable para el proxy o stub del componente, de manera que no sea problema añadir más variables y mantener el estado de la información en el cliente.

Al aplicarse un diseño de caché al cliente, a la firma de la función (el nombre del método, los tipos de datos de sus parámetros y el valor de regreso) se le agregan parámetros por referencia que inicializan el estado al principio de la invocación del método para ser regresados al cliente y ser asignados para un uso futuro. Un estado más extenso requiere de más parámetros en el método.

Sin embargo, este enfoque no cumple con una de las reglas más importantes de la programación orientada a objetos y del desarrollo de componentes: la encapsulación, la cual es información oculta que limita el acceso a los detalles internos de un objeto. Al no haber encapsulación, se termina con la capacidad del objeto para mantenerse dentro de una aplicación tradicional cliente/servidor en dos capas y dentro de una aplicación distribuida en *n* capas.

Cuando se necesita añadir y administrar otra variable, la aplicación debe ser reconstruida simultáneamente en el servidor y en el cliente, lo cual significa mayor pérdida de tiempo. Esta técnica es buena para una investigación simple y para el

desarrollo de aplicaciones, pero no es recomendable para un trabajo a gran escala o para una aplicación crítica de producción, que requiera alta disponibilidad.

Un cliente ligero Web evita alguno de los problemas relacionados con los cambios en el servidor y en el cliente, ya que la aplicación del cliente se almacena en un servidor y se descarga cada vez que se utiliza. Los clientes Web normalmente usan cachés de estado en un solo sentido utilizando una *cookie* que almacena el estado en un par nombre / valor dentro de un archivo transferido entre el cliente y el servidor Web.

Ventaja:

- Este diseño es fácil de entender e implementar.

Desventajas:

- Se requiere de firmas de funciones grandes para poder incluir información adicional.
- No respeta la encapsulación.
- El tráfico en la red se incrementa, lo cual puede requerir aumentar el ancho de banda.

II.3.8.2 Diseño del caché en la base de datos

Este tipo de diseño generalmente es la solución más realista. El principal beneficio de utilizar la base de datos para almacenar el estado de un componente es que es muy probable que el componente interactúe con la base de datos y accederla requiere de poco esfuerzo.

Existen inconvenientes técnicos para la implementación de este diseño. Algunas bases de datos, como Oracle no soportan tablas temporales al mismo grado que las de otros proveedores. Esto da como resultado tener que utilizar muchas tablas permanentes que contengan datos temporales. Además, puesto que hay mucha interacción con estas tablas temporales, es muy probable que un servidor que no esté bien configurado, genere cuellos de botella en la red durante el procesamiento de la aplicación.

Otra dificultad técnica es que las bases de datos relacionales generalmente no guardan registro de los objetos de datos consultados constantemente. Los objetos suelen ser muy complejos, por lo que dirigir los datos de la instancia a un renglón de una tabla podría no ser muy claro ni aconsejable. Las bases de datos orientadas a objetos (OODBMS) pueden ser una buena solución a este problema, debido a que almacenan fácilmente la instancia de un objeto, como si fuera campos de texto.

El servidor Jaguar puede trabajar con sólo una conexión caché, utilizando sólo una instancia para almacenar los datos principales.

Jaguar proporciona gran disponibilidad y balanceo de cargas. Estas características pueden ser aprovechadas por un grupo de dos o más servidores Jaguar que trabajan de manera conjunta para evitar fallas. Esta es una técnica de caché de datos para instancias de componentes stateless, que asegura que la información de los estados esté disponible para el componente sin depender del servidor Jaguar físico en el que se ejecuta el componente. El uso de esta técnica hace posible añadir nuevos servidores Jaguar al grupo, lo cual incrementa la escalabilidad de la aplicación sin implicar mayor trabajo.

Una de las preocupaciones al decidir usar un diseño de caché de base de datos es la manera en que se clasifican los datos. Este problema lo resuelve Jaguar de la siguiente manera. Cuando un cliente establece una conexión con el servidor Jaguar, éste se responsabiliza de la validación de los usuarios (asegurándose que los usuarios son quiénes dicen ser). Una vez validados, Jaguar asigna una contraseña de identidad (ID) a la sesión del cliente, la cual está estadísticamente garantizada para ser única, inclusive entre diferentes servidores Jaguar.

Combinando el identificador de la sesión con el nombre del usuario, se forma una llave primaria. Esta técnica funciona bien cuando el cliente se conecta constantemente. Sin embargo, cuando no es así y se utiliza el poleo de conexiones, esta técnica no es la mejor, ya que muchos recursos se desperdician innecesariamente, por lo cual se debe usar otro algoritmo que genere el identificador de la sesión. Por ejemplo, una implementación sencilla puede generar números aleatorios y concatenarlos a la clave de acceso del usuario.

Cuando se usan bases de datos para guardar la información de los componentes *stateless,* es necesario diseñar un mecanismo que elimine automáticamente los datos cuando ya no sean necesarios. Una forma de hacerlo, es utilizar un objeto de servicio que borre los datos basándose en la fecha de modificación para determinar si la duración de tiempo predefinida ya ha pasado y los datos se pueden borrar de manera segura.

Ventajas:

- Muchos componentes ya cuentan con conectividad a la base de datos, de modo que se requiere de poco o ningún trabajo extra de programación.

- Este diseño puede ser implementado fácilmente en un caché secundario de base de datos con un Adaptative Server Anywhere o con una base de datos similar localizada directamente en el servidor Jaguar.

- Este diseño se adapta a un grupo de servidores Jaguar sin ningún cambio en la codificación de los componentes.

- Jaguar puede proporcionar la mayoría de las veces, el ID de la sesión como llave primaria.

Desventajas:

- Se requiere de un buen servidor de base de datos, con controladores múltiples de discos, para reducir los cuellos de botella al mínimo posible.
- Puede haber restricciones, debido al almacenamiento de datos temporales en tablas permanentes (en ciertos sistemas de bases de datos).
- El tráfico en la red entre servidores se incrementa, aunque usualmente el tráfico está en un ancho de banda mayor que el de las conexiones de los clientes.
- Puede ser difícil localizar los datos accedidos constantemente de los componentes en un esquema relacional.

II.3.8.3 Diseño del caché para archivos planos

No es recomendable utilizar este tipo de diseño, ya que aunque es sencillo de entender y codificar, se debe poner atención en el escenario global del problema. Primero se tienen que escribir todas las rutinas de manejo, incluyendo el diseño de la plantilla del archivo (como .ini, delimitado por comas o por tabuladores). También hay que considerar la forma de coordinar el acceso al archivo por los componentes que solicitan datos simultáneamente, asumiendo que se utiliza sólo un archivo para manejar el estado de los componentes. Una opción es utilizar varios archivos, uno por cada usuario.

La utilización de archivos para almacenar datos *stateful* entre invocaciones al método es aceptable para un diseño con un solo servidor Jaguar. Sin embargo, para aprovechar la capacidad de Jaguar, es recomendable utilizar un diseño de caché de base de datos en lugar de uno de archivo plano. Es decir, que una base de datos está específicamente diseñada para manejar accesos concurrentes y decidir quien puede accederla. Además, los grupos Jaguar pueden dirigirse a una sola base de datos para poder acceder a los datos *stateful* necesarios; lo cual es más sencillo que mantener muchos archivos entre varias máquinas.

Si es necesario utilizar un diseño de caché de archivos, es conveniente implementar un archivo plano por usuario para evitar problemas al accederlo. De esta forma muchas instancias de un componente pueden ser leídas simultáneamente y escritas en su propio archivo, al cual se le deberá generar un nombre único que será devuelto al cliente, en donde deberá hacerse el caché para futuras llamadas al método. Para realizar lo anterior, se tienen las siguientes opciones:

- Si exclusivamente se están utilizando servidores Windows NT, entonces se pueden usar las funciones de la API de Win32 GetTempFileName() y GetTempDirectory().

- Se puede utilizar el identificador de la sesión como nombre del archivo, con lo cual no hay necesidad de regresar nada al caché del cliente.

- Si los componentes fueron desarrollados con Java, existe la opción de serialización de objetos, la cual convierte un objeto Java en un tipo de dato *blob* de un bit. Después de la conversión, se puede enviar a cualquier lugar, incluso a un archivo. Cuando se vuelven a necesitan los objetos, el *blob* es deserializado para convertirse otra vez en un objeto Java. Java maneja la mayoría de los detalles durante esta conversión, y de hecho, cada componente EJB soporta la interfaz java.lang.Serilizable como parte de la especificación EJB.

Cuando se utilizan archivos planos para almacenar datos *stateful* desde un componente *stateless*, es necesario diseñar un mecanismo que elimine automáticamente los archivos no utilizados. Sin importar que tipo de implementación de almacenamiento de archivos, la mejor opción es diseñar un servicio de Jaguar que opere tan frecuentemente como sea necesario. Dentro de este servicio se hace una lista de todos los archivos que contienen a los archivos *stateful* en un directorio, se seleccionan aquellos que deban ser eliminados de acuerdo a su última fecha de modificación y se determina si ha pasado el tiempo suficiente para borrarlos con seguridad.

Ventajas:

- El ancho de banda de la red no aumenta.

- Los sistemas operativos soportado por Jaguar tienen rutinas de archivo fáciles de usar.

- Jaguar proporciona un nombre único de archivo para clientes que no sean Web, el cual es el identificador de la sesión.

Desventajas:

- Jaguar resuelve automáticamente las fallas en los componentes *stateless* sin permitir que el desarrollador las controle.

- Un diseño basado en un solo archivo produce un cuello de botella en el sistema, por lo cual la instancia del componente del cliente se queda esperando en una cola para poder acceder al archivo y leer y/o escribir datos.

- El desarrollador debe diseñar un sistema que elimine los archivos después de cierto tiempo.

II.3.8.4 Diseño del caché de componentes compartidos

Dependiendo del lenguaje utilizado para implementar un componente compartido, serán sus capacidades. El componente puede manejar el caché para datos *stateful* sin importar las técnicas utilizadas.

Una ventaja de este tipo de componentes, es que el objeto está instalado y almacenado en RAM, lo cual permite una acceso más rápido a la base de datos local o a un archivo. La desventaja es su capacidad limitada para repartir los datos *stateful* del componente compartido en varios servidores Jaguar dentro de un grupo. Generalmente se piensa en que el componente compartido se encuentre en un solo servidor, por lo que no es necesario repartir sus valores entre varios servidores, lo cual es una visión muy pobre del diseño, ya que el sistema no aprovecha ni el balanceo de cargas ni la alta disponibilidad. La combinación de este diseño con la técnica de base de datos ayuda a solucionar este problema.

Desarrollar un componente de cualquier tipo en un solo servidor Jaguar, posibilita que existan fallas, y diseñar una aplicación de alta disponibilidad, es importante evitarlas. El diseño debe implementar un componente en varios servidores, como se ilustra en la figura siguiente. De tal forma, que si el servidor tuviera algún problema, otro servidor se encargaría de la carga de trabajo, aunque le afectara un poco en su funcionamiento.

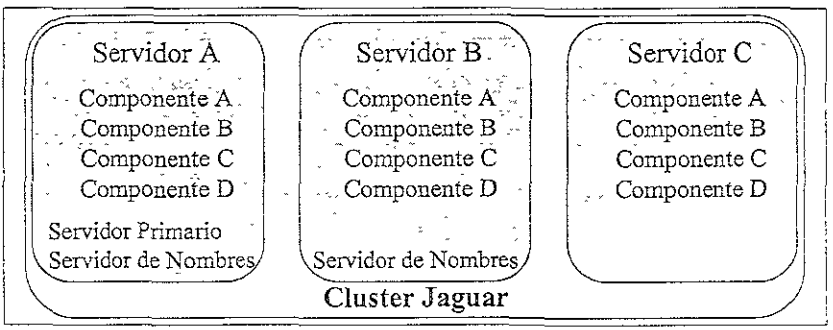


Fig. II.7 Distribución de los componentes en un grupo de servidores

Ventajas:

- El tráfico en la red no se incrementa.
- Este diseño puede utilizar en el caché el identificador de la sesión como contraseña primaria.

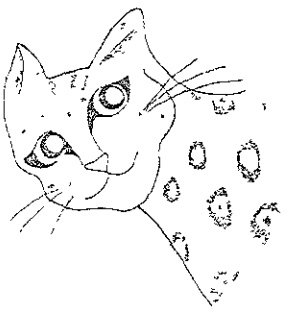
Desventajas:

- La capacidad de Jaguar para resolver fallas se ve muy comprometida en componentes que requieren comunicarse con un objeto compartido diseñado para operar en un solo servidor.
- Las operaciones pequeñas se ejecutan junto con los objetos compartidos (en caso de haber sido desarrollado en PowerBuilder), dando como resultado una cola *FIFO* y un cuello de botella en el sistema.

En el siguiente capítulo se hablará de las características de Jaguar y de su funcionamiento.

"Los grandes conocimientos
engendran las grandes dudas."

Aristóteles.



Capítulo III Servidor Jaguar CTS

Actualmente, muchas empresas grandes y pequeñas realizan negocios a través de Internet, para lo cual son necesarias las aplicaciones con contenido dinámico, acceso seguro a bases de datos y procesamiento de transacciones a gran escala.

Jaguar es un servidor de transacciones de componentes diseñado para construir aplicaciones multi-capas de negocios. Cuenta con la tecnología necesaria para desarrollar con facilidad y eficiencia este tipo de aplicaciones en la Web.

Como se dijo anteriormente, combina las mejores características de un ORB y de un monitor de procesamiento de transacciones basado en componentes. Además, su núcleo multi-hilos mejora el ambiente del componente en tiempo de ejecución, haciendo que las aplicaciones sean más rápidas. Trabaja con todos los modelos líderes de componentes, simplifica la instalación y configuración de las aplicaciones basadas en transacciones, y está orientado al cómputo abierto, por lo que se puede invertir en software y hardware sin preocuparse por los proveedores.

Jaguar pertenece a la compañía Sybase, que liberó su primer Conjunto de Desarrollo de Software (SDK) en febrero de 1997. Proporciona un marco para el desarrollo de la lógica en la capa intermedia de aplicaciones distribuidas basadas en componentes, para simplificar la creación y administración de las aplicaciones de Internet que atienden simultáneamente a muchos usuarios. Además proporciona un manejo eficiente de las sesiones cliente, de la seguridad, de los hilos, de las conexiones a bases de datos y de las transacciones; y su escalabilidad e independencia de plataformas permiten desarrollar la aplicación en máquinas de un solo procesador y posteriormente llevarlas a uno multi-procesador.

Las características generales de Jaguar son:

- Mecanismo de ejecución escalable y de plataforma independiente.
- Conectividad rápida entre todas las capas.
- Desarrollo basado en componentes, con soporte para los modelos líderes como: Active X, JavaBeans, C++ y CORBA.
- Seguridad en Internet, como encripción SSL, autorizaciones y listas de control de acceso a aplicaciones en capas.
- Manejo flexible de transacciones incluyendo el procesamiento de transacciones tradicionales síncronas y asíncronas.

- Administración gráfica con Jaguar Manager, incluyendo al navegador de interfases de componentes, a la seguridad basada en perfiles para restringir las conexiones del cliente y la invocación de los componentes, y al monitoreo en tiempo de ejecución.

- Manejo del ciclo de vida de los componentes.

- Cachés de conexión para reutilizar las conexiones a bases de datos.

- Manejo de transacciones de la aplicación.

- Manejo de hilos para simplificar el uso de datos y los recursos compartidos.

- Soporte de conjuntos de resultados con estructura de datos tabulares.

III.1 Funcionamiento

A continuación se describen las características del funcionamiento de Jaguar, como son su mecanismo de ejecución, núcleo, manejo de sesiones y conexiones, y administración.

III.1.1 Mecanismo de ejecución

Está diseñado para proporcionar un funcionamiento predecible y escalable para muchos usuarios. Sus características son:

- Un núcleo (*kernel*) escalable, multi-hilos y multi-procesador.

- Manejo avanzado de sesiones y conexiones.

- Sistema de monitoreo y administración basado en un navegador.

III.1.1.1 Núcleo

Un punto clave del buen funcionamiento de Jaguar, es su núcleo multi-hilos y multi-procesador, el cual tiene las siguientes caracterísitcas:

- Proceso multi-hilos: hace posible que los hilos de los sistemas operativos nativos estén disponibles.

- Construcción de hilos (*built-in threading*): para plataformas sin implementación de hilos nativos.

- Manejo de hilos para componentes específicos.

- Escalabilidad del multi-procesador.

- Variedad de parámetros de ajuste para un buen funcionamiento en diferentes plataformas y cargas de trabajo.

El núcleo del Jaguar oculta la complejidad del manejo de hilos, y del bloqueo y administración de la memoria. Esto es importante, considerando lo difícil que es escribir aplicaciones servidor para muchos usuarios.

Muchos sistemas, incluyendo a los ORBs y a los servidores de aplicaciones Web, dejan que los desarrolladores se imaginen el manejo de hilos, el bloqueo y la administración de memoria. Estas funciones deben realizarse de forma particular para cada plataforma de sistema operativo.

III.1.1.2 Manejo de sesiones y conexiones

En comparación con los sistemas tradicionales cliente/servidor, o con los *mainframes*, las aplicaciones *WebOLTP*, particularmente aquellas para Internet o extranet, están hechas para muchos usuarios y cuentan con un manejo eficiente de sesiones y conexiones.

La administración de sesiones se refiere al manejo de la comunicación entre el navegador y el servidor de transacciones; mientras que la administración de conexiones se refiere a la comunicación entre el servidor de transacciones y el *back-end* de las bases de datos.

Juntos, los manejadores de sesiones y de conexiones de Jaguar, pueden convertir muchas sesiones de los navegadores, en pocas conexiones al DBMS, mejorando la escalabilidad del sistema y asegurando tiempos de respuesta consistentes para cargas de trabajo no previsibles.

El manejador de sesiones administra transparentemente las comunicaciones entre el navegador y Jaguar. A diferencia de la mayoría de los sistemas, las sesiones de Jaguar son independientes del protocolo, lo cual permite, por ejemplo, que una sesión empiece usando el protocolo HTTP, común de la red, y después lo cambie al protocolo de flujo de datos de alta velocidad **TDS** (*Tabular Data Stream*). El poder manejar estas interacciones dentro de una sola sesión, permite que el servidor conserve el estado y la información de un usuario específico.

El manejador de conexiones administra transparentemente la comunicación entre Jaguar y la base de datos. Una característica clave es su *poleo* de conexiones, con el

cual el administrador puede asignar anticipadamente un grupo de conexiones a uno o más DBMSs. Tales conexiones pueden compartirse entre varios usuarios y reducir la carga total en los DBMSs finales, debido a que controlan el número de conexiones simultáneas y a que eliminan los costos de instalación de la conexión por usuario. Además, el manejador de conexiones se relaciona con las sesiones de Jaguar y con las transacciones del DBMS.

Estas características incrementan la escalabilidad y reducen los tiempos de respuesta del usuario final.

III.1.1.3 Administración y monitoreo

Jaguar tiene su propio sistema de administración y monitoreo llamado Jaguar Manager, el cual es fácil de usar y está escrito por completo en Java. Puede utilizarse en navegadores o en cualquier plataforma que soporte Java, y puede hacer lo siguiente:

- Conectarse y desconectarse del servidor Jaguar
- Configurar la memoria, y el número de sesiones y de conexiones.
- Monitorear el funcionamiento del servidor y la ejecución de los *servelets.*

Debido a que los límites entre el desarrollo, las pruebas, el despliegue y la administración de funciones no son muy claras, el sistema de administración de Jaguar tiene integrado un administrador de paquetes, en el cual los desarrolladores pueden desplegar dinámicamente un nuevo componente e inmediatamente monitorear su funcionamiento, por lo cual es posible usar una sola herramienta para administrar todo el ciclo de vida de la aplicación. A continuación se presenta la ventana de Jaguar Manager.

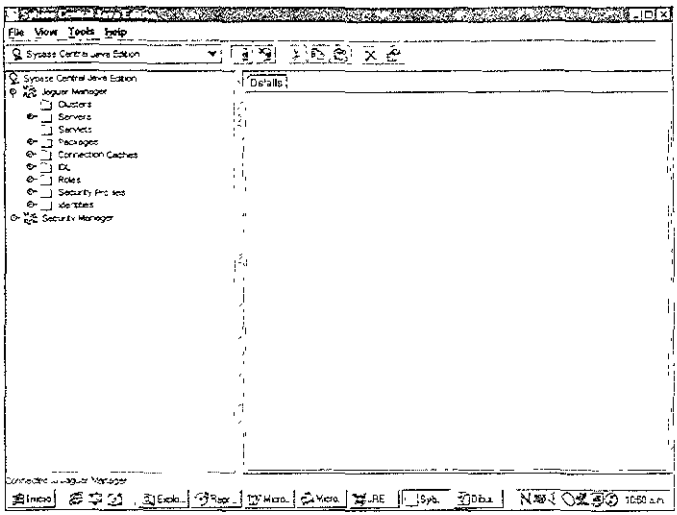


Fig. III.1 Jaguar Manager

III.1.2 Conexiones de Jaguar CTS

En un ambiente multi-capas, la velocidad de las comunicaciones muchas veces determina más que ningún otro factor, el tiempo de respuesta del usuario final. Jaguar proporciona una comunicación rápida y una conectividad optimizada para los navegadores y los contenedores de las bases de datos finales. Sus características de conectividad son:

- Soporte generalizado multi-protocolo para navegadores y otros clientes de la red. Los protocolos soportados son HTTP, TDS e IIOP CORBA.
- Conexión a los principales DBMSs, como SQL Server y SQL Anywhere de Sybase, Oracle 7.x, MS SQL Server a través de los estándares ODBC, JDBC, y Sybase Open Client.
- Conexión a *mainframes* y otras fuentes de datos, disponibles a través de los productos para acceso a datos de Enterprise Connect de Sybase.
- Conectividad nativa y de alta velocidad de Java para *applets* y *servlets*.
- Alta velocidad de flujos de conjuntos de resultados entre todas las capas.
- *Tunneling* eficiente de HTTP, en caso de necesitarlo por compatibilidad con el *firewall*.

III.1.2.1 Soporte multi-protocolo

Algunos proveedores apoyan al protocolo HTTPS y otros al IIP, o al DCOM. Jaguar protege a los desarrolladores de la complejidad e incertidumbre, mediante un soporte generalizado multi-protocolo. Un servidor Jaguar puede utilizar varios protocolos simultáneamente y combinarlos en una sola sesión. Esto hace posible que soporte tanto protocolos estándares, como flujos de conjuntos de resultados a través del protocolo de alta velocidad TDS de Sybase. También se pueden añadir fácilmente, nuevos protocolos para satisfacer los requerimientos del usuario o para cumplir con nuevos estándares. En cualquier caso, Jaguar hace transparente el protocolo subyacente, al desarrollador.

III.1.2.2 Conexión a DBMSs

Jaguar proporciona conexión hacia la mayoría de las fuente de datos finales. Entre los DBMS más importantes están los servidores SQL de Sybase, de Oracle, de Microsoft y de Informix. La conexión a éstos, se realiza mediante un ODBC estándar e interfases JDBC. La conexión a más de veinte fuentes de datos de *mainframes* y de otras fuentes de datos heredadas, como DB2 de IBM, IMS y VSAM, se realiza mediante los productos de acceso a datos del Enterprise Connect de Sybase. Las conexiones y las transacciones de las bases de datos son administradas por los manejadores de conexiones y de transacciones de Jaguar.

III.1.2.3 Conexión a Java

Debido a la importancia de Java en muchos desarrollos de sistemas para la WebOLTP, Jaguar proporciona conexiones muy veloces entre los *applets,* los *servlets* y los DBMSs finales. JDBC es un manejador estándar que comunica a las aplicaciones de Java con las bases de datos.

Los manejadores que no son para Java, requieren de instalaciones y configuraciones que impiden aprovechar los beneficios que dan las soluciones basadas en Java. Además, éstos manejadores suelen tener un funcionamiento deficiente, debido a que requieren de, al menos una etapa adicional de procesamiento y frecuentemente de trabajo adicional de la red, así como de conversiones del protocolo.

En contraste, Jaguar utiliza el manejador JDBC, conocido como **jConnect**, que está escrito completamente en Java, además es rápido, pequeño (<200 bytes de código) y soporta comunicaciones entre *applets* y *servlets*, y entre *servlets* y DBMSs.

III.1.3 Estándares

En el mercado creciente de Internet, los estándares aseguran la interoperabilidad y protegen la inversión. Jaguar ofrece la libertad de escoger las herramientas para el desarrollo del *front-end*, las bases de datos, y las plataformas que convengan más a las necesidades del negocio. También soporta varios estándares de servidores de

transacciones, como Java/JavaBeans, EJB, y los servicios de transacciones Java (JTS); y es compatible con las APIs del servidor de transacciones de Microsoft (MTS).

III.1.4 Soporte de despliegue

Para simplificar el despliegue de una aplicación, Jaguar Manager define las siguientes unidades de aplicaciones de las capas intermedias:

- **Servidor:** representa un proceso Jaguar en tiempo de ejecución. Cada servidor tiene sus propias direcciones de red para las conexiones de la sesión cliente y para las conexiones HTTP. Todos los servidores comparten el mismo almacén de configuración en una máquina host. Para administrar, es posible conectarse a cualquier servidor en la máquina host para configurar otros servidores del mismo host.

- **Paquete:** Organiza a los componentes dentro de unidades seguras que pueden ser fácilmente desplegadas a otro servidor Jaguar. Se pueden exportar o guardar como archivos JAR, que contienen la definición de sus componentes y de cualquier otro archivo de soporte (como código fuente y archivos del cliente) especificado. Los archivos exportados de un servidor se pueden importar a otro. Los perfiles de seguridad del paquete controlan el acceso a sus componentes.

- **Componente:** La definición de un componente en Jaguar Manager consiste de la firma de sus métodos y de otras propiedades, como el tipo de componente, el soporte de transacciones, el modelo de hilos y el nombre de la clase Java o de la librería ejecutable que lo implementa.

Antes de que una aplicación cliente pueda ejecutar un componente, éste debe estar instalado en un paquete de Jaguar, el cual a su vez debe estar instalado en el servidor al que se conecta el cliente.

III.1.5 Monitoreo en tiempo de ejecución

Se puede usar el Jaguar Manager para conectarse al servidor de Jaguar y ver los mensajes del archivo de registro (log) de ejecución o los de HTTP. También se puede obtener el estado de los clientes que tienen sesiones activas, así como el de los componentes, las transacciones y los cachés de conexión.

III.1.6 Soporte de modelos multi-componentes

Jaguar fue diseñado para soportar un modelo general multi-componentes. Esto hace posible que un servidor Jaguar ejecute simultáneamente, servlets construidos de acuerdo a cualquier modelo de componentes o de objetos de uso común, como ActiveX, Java y JavaBeans, C/C++ o IDL CORBA.

Los mismos modelos de componentes también son soportados por los applets. Jaguar

les permite solicitar y regresar datos transparentemente desde los servlets o componentes de Jaguar, incluso si éstos hubieran sido construidos originalmente usando diferentes modelos de componentes. Por ejemplo, un applet de Java podría invocar tanto a servlets de ActiveX, como de C++ nativo, que se ejecutan en la misma sesión de Jaguar.

III.1.7 Integración de componentes de terceros

Debido a que Jaguar soporta modelos estándares, los componentes de terceros, independientes de la plataforma, pueden trabajar en él. Entre los componentes disponibles están los siguientes:

- ActiveX y Java para hojas de cálculo, correctores de ortografía y herramientas de diagramación.
- Servicios completos para comercio electrónico y mensajería.
- Aplicaciones C/C++ ya existentes.

III.1.8 Sesión cliente y manejo del ciclo de vida de los componentes

Los clientes establecen sus sesiones con el servidor Jaguar, por medio de los *stubs* y *proxies* que llaman a los métodos de los componentes. El ciclo de vida de un componente determina la forma en que las instancias se colocan, se ligan a las sesiones cliente y se destruyen. Jaguar administra las sesiones del cliente y el ciclo de vida del componente sin necesidad de que el desarrollador tenga un conocimiento especializado en estos temas.

En la siguiente figura se muestra el ciclo de vida de un componente. Al invocarlo, se crea la instancia de la clase, la cual se activa, se liga al cliente y se ejecuta el método que fue llamado. Si la propiedad de desactivación automática está activada se llevará a cabo el poleo de la instancia, si no, dependiendo de la primitiva terminará su tarea y preguntará si es reciclable, es decir, si entra al pool o se destruye.

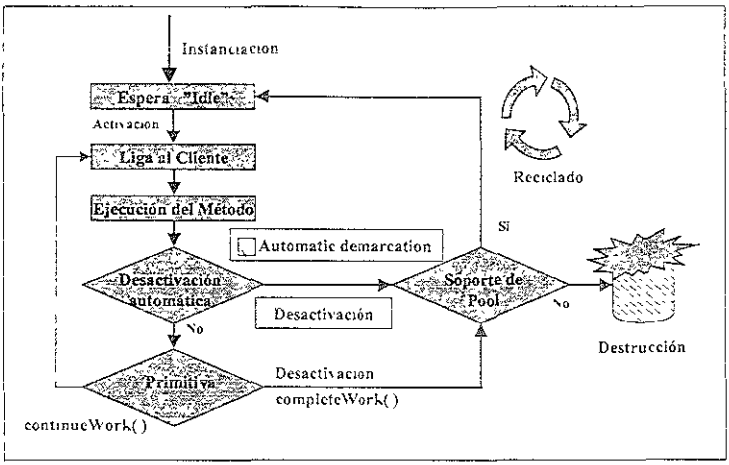


Fig. III.2 Ciclo de vida de un componente

III.1.9 Procesamiento del conjunto de resultados

Uno de los puntos fuertes de los sistemas cliente/servidor es el buen manejo del procesamiento de los conjuntos de resultados. En un sistema cliente/servidor de dos capas, para presentar y procesar los resultados de una base de datos, el desarrollador simplemente dirige el control de los datos (DataWindow en Power Builder) a un esquema de base de datos. Entonces, la herramienta automáticamente genera la lógica de presentación necesaria para desplegar los resultados.

En la mayoría de los sistemas multi-capas primero se examina la lógica del negocio de las capas intermedias para determinar que resultado será devuelto, y después se codifica el procesamiento de los conjuntos de resultados y de la presentación gráfica de éstos.

Jaguar resuelve este problema, con el uso de APIs que procesan los conjuntos de resultados y que están disponibles para los applets, permitiendo especificar el resultado que devuelve un componente. De esta manera, los controles de enlace de datos pueden generar automáticamente la lógica de presentación en los clientes ligeros.

III.2 Manejo de transacciones

Jaguar ofrece un manejo flexible de transacciones con soporte tanto para procesamiento de transacciones tradicionales síncronas y asíncronas basadas en colas, como se ve en la siguiente figura.

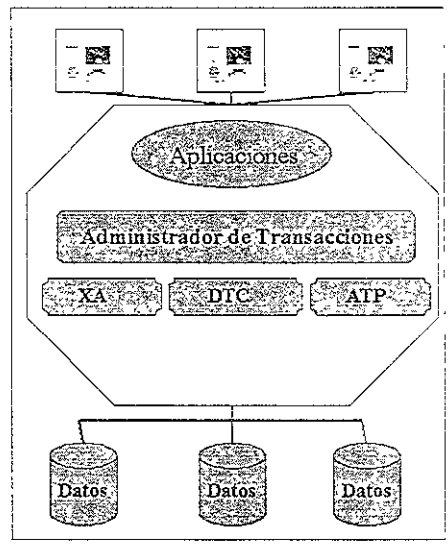


Fig. III.3 Manejo de transacciones de Jaguar CTS

El manejo de transacciones es abierto, ya que soporta muchos coordinadores de transacciones y de DBMSs. Algunas de sus características son:

- Convierte las transacciones en atributos al tiempo de despliegue.
- El manejo de transacciones es implícito.
- Automáticamente los componentes forman parte de las transacciones y pueden ser marcados como transaccionales en el Jaguar Manager.
- Las transacciones trabajan sólo con conexiones obtenidas desde el manejador de conexiones de Jaguar.

III.2.1 Manejo de transacciones síncronas

Jaguar oculta prácticamente toda la complejidad del manejo y la coordinación de las transacciones, ya que las ejecuta implícitamente. Durante el desarrollo, sólo es necesario especificar si un componente es o no transaccional, y durante la ejecución, los límites de la transacción se administrarán automáticamente, asegurando la consistencia transaccional en todos los componentes y DBMSs subyacentes. Cuando se requiere coordinar las actualizaciones de los datos en varios DBMSs, Jaguar, automática y transparentemente invoca al DTC de Microsoft o a un coordinador de transacciones XA.

III.2.2 Manejo de transacciones asíncronas

Aunque las transacciones sincrónicas son apropiadas para la mayoría de los sistemas, muchas aplicaciones WebOLTP generan transacciones físicas a través de sistemas anteriores y nuevos. Por ejemplo, un sistema puede recibir órdenes directamente de los consumidores en la Internet. Cuando la orden es recibida, genera transacciones en el sistema de envíos y en el sistema de facturas, como se ve en la figura siguiente.

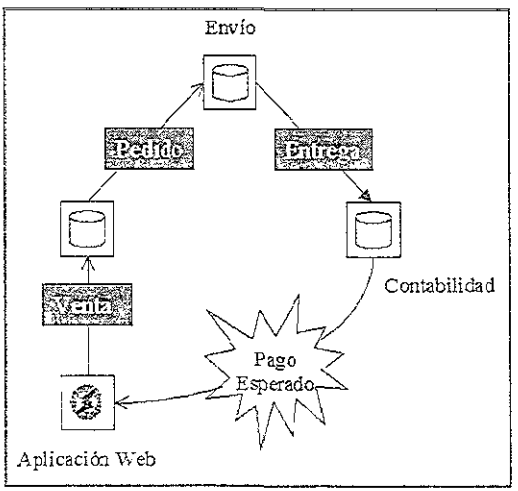


Fig. III.4 Ejemplo de una orden

Cada paso tiene la garantía de ser completado, pero solamente si el consumidor está dispuesto a esperar. Es impráctico obligar al usuario a esperar hasta que se completen todos los pasos a través de varios sistemas que no están disponibles. Para resolver este problema, Jaguar cuenta con el procedimiento de "disparo y olvido" y con su servicio de encolamiento de bases de datos (dbQ).

Con el dbQ, el usuario sólo tiene que pedir su orden mediante el sistema de recepción de órdenes, ejecutarla, e irse. El dbQ asegura que los sistemas de envío y contabilidad de la nueva orden sean notificados, mediante una mensajería confiable. A diferencia de los sistemas de mensajes generales, el dbQ emplea una base de datos con tecnología de colas para asegurar que las actualizaciones realizadas por los DBMSs en el sistema de orden/entrada, así como el mensaje para el sistema de envío, sean ejecutadas o desechas.

Las características principales del dbQ son:

- Colas residentes en la base de datos para asegurar la integridad de las transacciones, mejor funcionamiento y menos administración.
- Soporte para componentes ActiveX, JavaBeans y C++.
- Entrega flexible de mensajes con *fan-in/fan-out*, prioridades y notificación de envío.
- Soporte para varios DBMSs como SQL Server y SQL Anywhere de Sybase, Microsoft SQL Server y Oracle.
- Entrega confiable de mensajes a otros sistemas de mensajes, como MQseries de IBM.

III.2.3 Procesamiento de Transacciones

Una transacción Jaguar es aquella cuyos límites y resultados son determinados por Jaguar. En los componentes de tipo transacción, el manejador se asegura de que la dbQ se ejecute como parte de la transacción, en la cual varios componentes pueden participar, mientras el manejador cuida del funcionamiento de la base de datos, procesando las transacciones participantes.

III.2.3.1 Funcionamiento de las Transacciones

Cuando un componente es de tipo transacción y usa el manejo de conexiones de Jaguar, las ordenes enviadas en una conexión a las base de datos de la tercer capa se ejecutan automáticamente como parte de una transacción. Los métodos del componente pueden llamar a las primitivas de estado de las transacciones para determinar si Jaguar ejecuta o deshace la transacción actual.

El ciclo de vida de un componente está fuertemente integrado al modelo de transacciones de Jaguar. Las instancias del componente que participan en una transacción se desactivan hasta que la transacción termina o hasta que el componente indica que su participación en la transacción ha finalizado, es decir, que su trabajo está hecho y listo para ejecutarse o deshacerse. El tiempo que permanece una instancia en estado activo, es exactamente la duración de su participación en la

transacción.

Un beneficio de usar las transacciones para agrupar las actualizaciones a la base de datos es poder codificar fácilmente los métodos en un solo componente para implementar transacciones que funcionen en una misma fuente de datos. Sin embargo, estos métodos también pueden ser ejecutados por otro componente, el cual, por sí mismo define una transacción. En esta situación, la recuperación de errores se vuelve difícil. Por ejemplo, considerando el siguiente escenario en el que el componente **Inscripcion** llama a los componentes **Registro** y **Factura** :

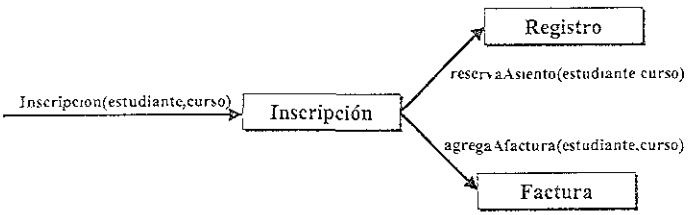


Fig. III.5 Ejemplo de Inscripción

El método **Inscripcion.inscribe()** llama a los métodos de los componentes **Registro** y **FacturaEstudiante**:

- **Registro.reservaAsiento()** revisa que un asiento esté disponible. Si es así, decrementa el conteo de asientos disponibles y agrega al estudiante en la lista de inscripción del curso. Si no hay asientos disponibles, **reservaAsiento()** regresa un error.

- **FacturaEstudiante.agregaAFactura()** revisa que el estudiante tenga crédito. Si es así, **agregaAFactura()** agrega el costo del curso a la factura de ese semestre del estudiante. Si el estudiante tiene un problema de crédito, **agregaAFactura()** regresa un error.

Las actualizaciones en la base de datos, hechas por los componentes **Registro** y **FacturaEstudiante** deben ejecutarse las dos o ninguna. Si no se puede hacer la factura para el estudiante, los asientos disponibles del curso no deben cambiar. Para manejar este caso, se puede escribir la lógica que deshaga los cambios en el método **inscribir()**, para lo cual se necesita el método **AsientoNoReservado()** en **Registro**. De cualquier forma, mientras más componentes se agreguen, la lógica que se necesita para deshacer rápidamente los cambios, se vuelve inmanejable. Es más fácil definir a todos los componentes participantes para que usen transacciones

Jaguar. Entonces, un error en cualquier componente, deshace los cambios hechos por otros componentes participantes antes de que ocurra el error.

Al definir componentes que usan transacciones, Jaguar se asegura de que el trabajo realizado por los componentes participantes en la transacción, ocurran como se planeó.

Para definir la participación de un componente en las transacciones, se debe:

- Escoger un coordinador de transacciones que maneje los flujos de transacciones que involucren a más de una conexión.
- Especificar el atributo de la transacción del componente para determinar si las instancias del componente participan en las transacciones.
- Codificar métodos que llamen a las primitivas de estado de la transacción Jaguar apropiada para reflejar el estado del trabajo que el componente ha realizado en la transacción.

III.2.3.2 Coordinadores de Transacciones

Todos los componentes instalados en un servidor Jaguar comparten el mismo coordinador transaccional. Las siguientes son opciones de coordinadores de transacciones:

- **Conexión compartida:** es un "pseudo coordinador" que se construye dentro de Jaguar. Con esta opción, los componentes participantes en una transacción comparten una sola conexión. Los datos de la aplicación deben estar en un servidor de datos y los componentes participantes en una transacción deben usar una conexión con el mismo nombre de usuario y password.
- **Coordinadores de transacciones distribuidas de Microsoft** (Distributed Transaction Coordinator, DTC): usa la ejecución en dos fases para coordinar transacciones entre múltiples bases de datos. DTC está disponible en Windows NT como parte del SQL Server 6.5 de Microsoft.

El default es la conexión compartida (Shared Connection). Para ver o cambiar el coordinador, se usa la opción de Propiedades del Servidor en el Jaguar Manager.

III.2.3.3 Atributos del Componente Transaccional

Un componente Jaguar tiene un atributo de transacción que le indica como participa en las transacciones. Este atributo puede ser visto y modificado con el Jaguar Manager en la pestaña "Transactions" de la ventana de propiedades del componente, y puede tomar los siguientes valores:

- **Transacción requerida (Required transaction):** El componente siempre se ejecuta en una transacción. Cuando la instancia de un componente es creada directamente por un cliente, empieza una nueva transacción. Si el componente A es activado por el componente B, y B se está ejecutando dentro de una transacción, entonces A se ejecuta dentro de ésta. Si B no se está ejecutando en una transacción, entonces A se ejecuta en una nueva.

- **Sin Transacción (Transaction not supported):** Los métodos del componente nunca se ejecutan como parte de una transacción. Si el componente es activado por otro que se está ejecutando dentro de alguna transacción, el trabajo de la nueva instancia se realiza fuera de la transacción existente.

Se debe especificar "Transacción requerida" para el componente, si cualquiera de lo siguiente es verdadero:

- Hay llamadas entre los métodos de los componentes y el trabajo hecho por los componentes llamados deben incluirse en una transacción.

- Los métodos de un componente podrían ser llamados por otro componente como parte de una transacción larga.

- Los métodos de un componente interactúan con más de una base de datos remota y las actualizaciones de diferentes bases de datos deben agruparse en la misma transacción.

En el escenario ilustrado en la figura anterior, a los tres componentes se les debe marcar la opción "Transacción Requerida" para lograr que el proceso para deshacer la transacción, sea correcto.

Se usa "Sin Transacción" para prevenir que las actualizaciones de la base de datos hechas por el componente, sean afectadas por los resultados de llamar a la transacción del componente.

Por ejemplo, si el componente registra las estadísticas del uso en una base de datos remota, puede ser marcado como "Sin Transacción" para asegurar que las actualizaciones del registro no sean afectadas si el componente es llamado por otro componente que deshace su transacción.

Después de que un cliente crea una instancia de un componente transaccional, la primer invocación al método inicia una transacción. Este componente es llamado **instancia raíz** de la transacción. Si el componente raíz invoca métodos de otros componentes transaccionales, éstos se unen a la transacción existente. El resultado de la transacción se determina por la forma en que los componentes participantes llaman a las primitivas de estado de la transacción.

III.2.3.4 Características transaccionales

Para manejar las transacciones de un componente, primero se deben configurar sus características transaccionales y después invocar a las primitivas de estado de éstas. A continuación se presentan las características de los componentes, dependiendo de su tipo y comportamiento.

Componentes que solicitan una transacción

Instanciado por	Comportamiento transaccional
Stub del cliente	Iniciar una nueva transacción
Otras llamadas entre componentes ejecutándose en una transacción	Se ejecuta en el componente que llama a la transacción
Otros componentes que no se ejecutan en una transacción	Inicia una nueva transacción

Componentes que soportan transacciones

Instanciado por	Comportamiento transaccional
Stub del cliente	No ejecutar en una transacción
Otras llamadas entre componentes ejecutándose en una transacción	Ejecutar en los componentes que llaman a la transacción
Otros componentes que no se ejecutan en una transacción	No ejecutar en una transacción

Componentes que solicitan una nueva transacción

Instanciado por	Comportamiento transaccional
Stub del cliente	Iniciar una nueva transacción
Otras llamadas entre componentes ejecutándose en una transacción	Iniciar una nueva transacción
Otros componentes que no se ejecutan en una transacción	Inicia una nueva transacción

Componentes que no soportan transacciones

Instanciado por	Comportamiento transaccional
Stub del cliente	No ejecutar en una transacción
Otras llamadas entre componentes ejecutándose en una transacción	No ejecutar en una transacción
Otros componentes que no se ejecutan en una transacción	No ejecutar en una transacción

En la siguiente ilustración se muestra el uso de las propiedades transaccionales en un ejemplo en el que se realizan operaciones bancarias en un cajero automático. Primero

se crea una instancia raíz, que inicia una transacción. Para las operaciones de débito y crédito, se llama a componentes que se ejecutan dentro de la transacción de la instancia raíz. En el caso de la operación de registro, se llama a un componente sin soporte de transacciones, es decir, que sus métodos no se ejecutan dentro de la transacción de la instancia raíz.

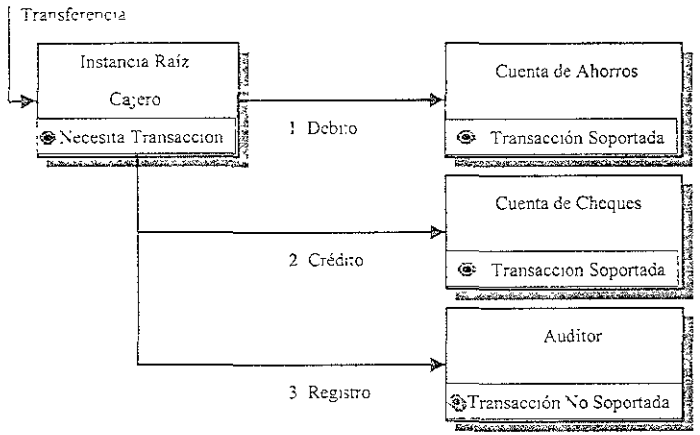


Fig. III.6 Ejemplo del uso de las características transaccionales

Jaguar proporciona primitivas de estado transaccional que los métodos pueden llamar para dirigir los resultados de la transacción actual. Todo modelo de componentes proporciona una interfaz con métodos para esas primitivas. La siguiente tabla describe como se invocan las primitivas de la transacción en cada modelo de componentes:

Primitivas de Transacción	Método de Java InstanceContext	Método ActiveX IObjectContext	Rutina C/C++
completeWork	completeWork()	setComplete()	jagCompleteWork
rollbackWork	rollbackWork()	setAbort()	jagRollbackWork
continueWork	continueWork()	enableCommit()	jagContinueWork
disallowCommit	disallowCommit()	disableCommit()	jagDisallowCommit
isInTransaction	isInTransaction()	isInTransaction()	jagInTransaction
isRollbackOnly	isRollbackOnly()	no soportada	jagIsRollbackOnly

Tabla III.1 Primitivas de Transacción Java, ActiveX y C/C++

Las siguientes primitivas finalizan la participación del componente en una transacción.

Ambos provocan que la instancia actual sea desactivada:

- **completeWork:** el componente termina su trabajo en la transacción actual y debe ser desactivado cuando el método regrese.

- **rollbackWork:** el componente no puede completar su trabajo. Señala a la transacción actual para ser deshecha y desactiva el componente cuando el método regresa.

Los siguientes métodos son usados para mantener el estado después de que el método regresa y retrasan la desactivación de la instancia del componente:

- **continueWork:** continúa la participación del componente en la transacción actual después de que el método regresa. Permite que la transacción se complete si el componente es desactivado. El comportamiento por default es llamar a una primitiva que no es de transacción.

- **disallowCommit:** continúa la participación del componente en la transacción actual después de que el método regresa, pero deshace la transacción si el componente es desactivado antes de llamar a otra primitiva.

Las siguientes primitivas se pueden usar para consultar el estado de la transacción (si existe) en donde el método se está ejecutando:

- **isInTransaction:** consulta si el método actual se está ejecutando en el contexto de una transacción.

- **isRollbackOnly:** consulta si la transacción actual está señalada para deshacerse o si todavía es válida.

Cualquier componente puede deshacer la transacción llamando a la primitiva **rollbackWork**. Los componentes Java lo pueden provocar al regresar una excepción que no se pueda manejar. Sólo la acción del componente raíz determina el momento en que Jaguar realiza la transacción, la cual se ejecuta por completo si pasa lo siguiente:

- El componente regresa con el estado **completeWork** y ningún componente participante tiene su estado en **disallowCommit**.

- El componente raíz es desactivado debido a su destrucción explícita desde el cliente y ningún componente tiene su estado en **disallowCommit**. (Una desconexión del cliente que no es precedida por una solicitud de destrucción explícita, siempre provoca que se deshaga la transacción.

Se pueden usar las primitivas de estado de la transacción en cualquier componente, sin que éste tenga que ser declarado de tipo transacción. La llamada a **completeWork** o a **rollbackWork** desde los métodos produce una desactivación temprana.

Ejemplo:

Las transacciones Jaguar son más útiles cuando la aplicación usa las llamadas entre componentes. Como en el ejemplo de la inscripción.

El pseudocódigo siguiente, muestra la lógica que se sigue para asegurar que el trabajo hecho por **Registro.reservaAsiento()** y **FacturaEstudiante.agregaAFactura()**, ocurra dentro de la misma transacción.

- **Registro.reservaAsiento()** debe revisar el número de asientos. Si hay lugar para un nuevo estudiante, el método agrega al estudiante, decrementa la cuenta de asientos disponibles y pone el estado en **completeWork**. Si no hay un lugar disponible, el método llama a **rollbackWork** para deshacer la transacción actual.

- **FacturaEstudiante.agregaAFactura()** verifica el crédito del estudiante. Si tiene crédito, el método agrega el costo a la factura del semestre y pone el estado **completeWork**, de otra forma, el método llama a **rollbackWork** para deshacer la transacción actual.

- **Inscripcion.inscribe()** primero llama a **Registro.reservaAsiento()**. Después de que **Registro.reservaAsiento()** regresa, el método revisa si la transacción todavía es válida, mediante el uso de la primitiva **isRollbackOnly**. Si la transacción es válida, el método llama a **FacturaEstudiante.agregaAFactura()**.

Para que el ejemplo funcione correctamente, estos tres componentes deben tener el atributo "Requires Transaction". Si **Registro** y **FacturaEstudiante** son de tipo transacción, pero **Inscripcion** no, entonces el trabajo hecho por

Registro.reservaAsiento() y **FacturaEstudiante.agregaAFactura()** estará en transacciones separadas. Estos métodos deben estar en la misma transacción para poder deshacerla correctamente.

III.3 Manejo de Conexiones

El manejador de conexiones controla a los cachés usados por los componentes para interactuar con distintos servidores y atender a muchos clientes utilizando pocas conexiones hacia los servidores de bases de datos. El manejador proporciona clases de Java y rutinas de C para obtener y liberar las conexiones, así como rutinas útiles para monitorear el uso del caché.

III.3.1 Caché de conexión

Un caché de conexión es una estructura interna de Jaguar que mantiene un poleo de conexiones disponibles para otro servidor. Dentro del caché, las conexiones pueden ser manejadores de conexión de librerías de cliente (apuntadores CS_CONNECTION) o manejadores de conexión ODBC (manejadores HDBC). Todas las conexiones del caché deben compartir nombre y clave de usuario comunes, conectarse al mismo servidor, y usar la misma librería de conectividad.

Debido a que el caché es una conexión a bases de datos que los componentes usan para interactuar con servidores de bases de datos, el servidor Jaguar puede atender a muchos clientes usando un número limitado de conexiones a las bases de datos. Es muy importante que cuando se use un caché de conexión, se obtengan y se liberen frecuentemente las conexiones para evitar que el servidor entre en un estado inconsistente.

Los componentes que usan transacciones, deben usar cachés para interactuar con bases de datos remotas, ya que mejoran su funcionamiento y escalabilidad, por las siguientes razones:

- El caché permite a las sesiones cliente compartir las conexiones abiertas previamente a bases de datos. Esto ahorra tiempo en la conexión a éstas.

- No se necesita crear una conexión para cada sesión cliente. Los clientes pueden compartir recursos.

Para obtener estos beneficios, el componente debe codificarse para usar un caché sólo cuando sea necesario y en otras ocasiones para liberar la conexión del caché. No se debe permitir que los componentes mantengan conexiones mientras esperan más datos del cliente.

El caché permite a los componentes Jaguar, compartir los poleos de conexiones preasignadas a un servidor de bases de datos remoto, evitando que cada instancia de un componente cree una conexión por separado. Los componentes que realizan transacciones deben usar una conexión desde un caché para interactuar con la base de datos remota.

Para componentes codificados en C, Jaguar soporta ODBC y caches de conexión de librerías cliente. Para componentes en Java, proporciona los cachés de conexión JDBC. Para cada modelo de componentes hay una interfaz para realizar el poleo de conexiones y se utiliza el Jaguar Manager para definir los cachés usados por la aplicación.

III.3.2 Manejo de conexiones Java

Los componentes Java pueden usar las clases del manejador de conexiones Java (Java Connection Manager, JCM) para trabajar con el caché de conexión. Estas clases manejan conexiones JDBC, como se ilustra en la siguiente figura.

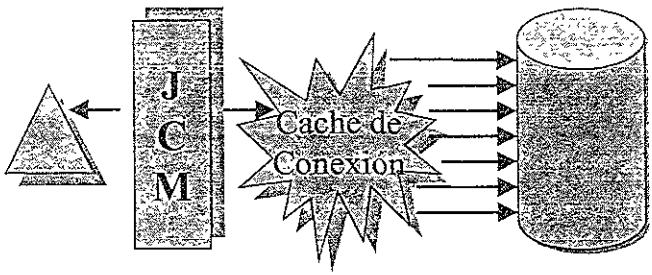


Fig. III.7 Caché de conexón para Java

Las clases JCM son:

- **com.sybase.jaguar.jcm.JCMCache**, representa un caché de conexión configurado, y proporciona métodos para manejar las conexiones en el caché.

- **com.sybase.jaguar.jcm.JCM**, proporciona acceso a caches de conexión JDBC definidos en el Jaguar Manager. **JCM** es un fabricante de instancias **JCMCache**.

III.3.3 Manejo de conexiones en componentes C y ActiveX

Los componentes ActiveX y C pueden llamar a las rutinas del manejador de conexiones para aprovechar el caché de conexión. Estas rutinas manejan los cachés de conexiones ODBC o de librerías de clientes. Las rutinas del manejador de conexiones tienen nombres que empiezan con **JagCm**.

En el archivo de cabecera *jagpublic.h* se declaran las rutinas del manejador de conexiones y las estructuras de datos. *jagpublic.h* se localiza en el subdirectorio *include* del directorio de instalación de Jaguar.

Antes de incluir a *jagpublic.h*, se deben incluir los archivos de cabecera requeridos por la librería de conectividad que se quiera usar.

III.3.4 Manejador de caché

El manejador de conexiones proporciona un manejador de control de caché CM_CACHE, que permite al código hacer referencia a un caché especifico. La mayoría de las rutinas del manejador de conexiones requieren la dirección de un manejador CM_CACHE como parámetro.

III.3.5 Propiedades de las bases de datos

El nombre del servidor de conexiones, el del usuario y el password, se fijan cuando se establece el caché. Sin embargo, otras propiedades de la conexión se pueden cambiar dinámicamente cuando se abre la conexión. Por ejemplo:

- **Opciones de conexión del lado del Servidor:** las llamadas **ct_options**, los comandos de lenguaje 'set' y las llamadas ODBC equivalentes, afectan la respuesta del servidor a los comandos enviados en la conexión.

- **Contexto de la base de datos:** distintos usuarios de un caché de conexión pueden usar diferentes bases de datos. Se puede evitar problemas cambiando explícitamente la base de datos cada vez que se use un caché.

- **Propiedades de conexión:** afectan el comportamiento de la conexión del lado del cliente.

Las siguientes son sugerencias para evitar problemas con los estados de conexión inconsistentes:

- Configurar cualquier opción y propiedad que necesite el código cuando se obtiene una conexión.

- Si el código comparte un caché con otras conexiones, las propiedades y las opciones modificadas deben volver a sus valores originales antes de liberar la

conexión.

- Si el código es el único usuario de un caché, no se necesita devolver las opciones y las propiedades a sus valores originales. Sin embargo, se tienen que reconfigurar las propiedades con sus valores originales cuando se vuelve a consiguir la conexión.

La siguiente figura muestra las propiedades de un caché de conexiones.

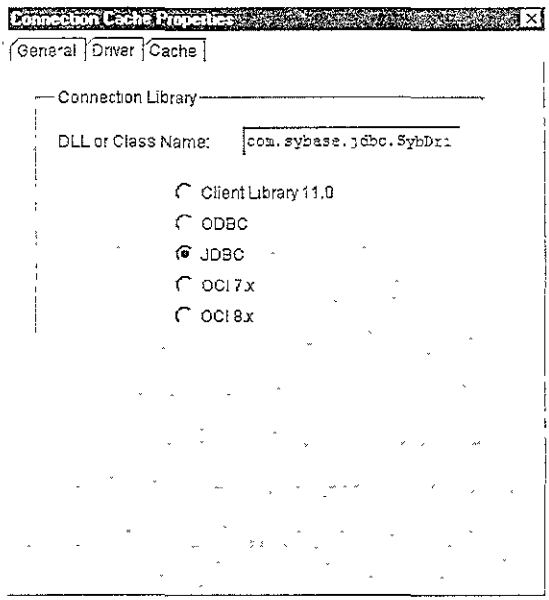


Fig. III.8 Propiedades del caché de conexión

A continuación se muestran los valores que pueden tomar las propiedades "nombre de la clase" y "nombre del servidor" del caché de conexión.

	Valor del nombre del servidor	Ejemplo
Cliente Library 11.0	Nombre del servidor desde SQL.INI (NT) Archivo de interfases (Solaris)	Ventas
ODBC	Nombre de la fuente de datos ODBC	LosLogos
JDBC 1.1	Fuente de datos URL	jdbc:sybase:Tds:ventas:5000 • jdbc:odbc:LosLogos

	DLL o nombre de la clase	Ejemplo
Cliente Library 11.0	CTLIB.DLL	libjct.dll (NT) libjct.so (Solaris)
ODBC	ODBC.DLL	odbc32.dll (NT) libodbc.so (Solaris)
JDBC 1.1	Manejador del nombre de clase	com:sybase:jdbc:SybDriver • sun:jdbc:JdbcOdbcDriver

III.3.6 Errores por resultados pendientes

Se debe tener cuidado de no liberar una conexión con resultados de comandos no procesados asociados a ésta. Cada vez que se necesite un comando que use un caché de conexión, los resultados del comando se deben procesar por completo antes de liberar la conexión para ser reutilizada. Una falla al procesar los resultados, provoca errores en el siguiente componente que use esa conexión. Nunca se debe liberar una conexión dentro de un caché diferente a aquel en donde fue creada.

III.4 Características de seguridad en hilos

Aunque este término ya ha sido utilizado en los capítulos anteriores, vale la pena profundizar en su concepto, para entender mejor el manejo de hilos que realiza Jaguar.

III.4.1 Definición de hilo

Un sistema operativo multi-tarea, como Windows 95/98 o Windows NT, parece poder ejecutar varias aplicaciones al mismo tiempo, pero en realidad sólo una aplicación puede usar el CPU en cierto momento. El procesamiento es dividido entre las aplicaciones, de forma que parece que se están ejecutando simultáneamente. Esto se realiza usando un mecanismo de rebanadas de tiempo (*time slicing*), en donde a cada proceso de la aplicación se le asigna un tiempo de CPU para ejecutarse. Cuando el tiempo asignado es alcanzado, el sistema operativo detiene la ejecución del proceso actual y empieza a ejecutar el siguiente proceso de acuerdo a un algoritmo calendarizador.

Una aplicación o programa, es llamado proceso. En un sistema operativo Windows de 32 bits, todo proceso es independiente de cualquier otro proceso que se está ejecutando en la máquina. Este tiene su propio espacio de direccionamiento de memoria y usa una pila de programa separada. Esto protege al proceso de sobrescribir la memoria de otro proceso o de comunicarse con otro proceso sin usar un protocolo de comunicación inter proceso, como OLE o DDE. Esto también protege a una aplicación de dar de baja al sistema por completo.

En realidad, un proceso está formado por uno o más hilos. La documentación de Windows NT define un hilo como la unidad más básica del sistema operativo, capaz de ejecutar instrucciones en el CPU. Un hilo representa un camino de ejecución y tiene su propio espacio de direcciones y su propia pila de programación. Todo proceso tiene al menos un hilo, el hilo primario, el cual es iniciado automáticamente con la aplicación.

Un programa común sólo tiene un camino de ejecución en un hilo primario, en el cual se ejecuta la parte *main* (aplicación consola) o *WinMain* (aplicaciones GUI de Windows) del programa. Al igual que una máquina puede ejecutar varios procesos, un proceso puede tener varios hilos ejecutándose dentro de él, y por lo tanto varios caminos de ejecución. A este tipo de aplicación se le llama multi-hilo.

En un sistema operativo multi-tarea, el núcleo es responsable de calendarizar el tiempo de CPU para todos los hilos en el sistema. Esto determina cuanto tiempo se ejecuta un hilo y cuando se volverá a ejecutar, y previene que una sola aplicación monopolice los recursos del sistema. Debido a que un hilo puede ser iniciado o detenido en cualquier momento por el sistema operativo, éste normalmente está a la mitad de la ejecución de la lógica de programación, cuando es interrumpido. Cuando un hilo es detenido, su estado actual necesita ser guardado para poder reanudar el procesamiento en donde se quedó, cuando llegue su próximo turno de usar el CPU. El estado o contexto de todo hilo es conservado por el sistema operativo en un objeto de contexto. En cualquier momento en el que un hilo es detenido, toda la información de los registros, la pila del programa y la memoria, son copiados a este objeto. Cuando el hilo es iniciado otra vez, toda esa información se vuelve a cargar antes de que la ejecución continúe. Esto es conocido como *context-switching* y es transparente para el hilo y para el desarrollador. Sin embargo, las tareas involucradas con el manejo de

hilos para una aplicación, utilizan recursos importantes del sistema y deben ser planeados cuidadosamente.

III.4.2 Características de seguridad de hilos en Jaguar

Debido a que Jaguar es un ambiente multi-hilos, las instancias de los componentes que comparten los recursos y los datos temporales, se deben codificar o configurar de tal forma que se evite la existencia de datos inconsistentes. Por ejemplo, si todas las instancias de un componente escriben en el mismo archivo, se deben seguir ciertos pasos para asegurar que tal archivo esté bloqueado antes de que alguna instancia le escriba, y desbloquearlo cuando se acabe de escribir. Si se permite escribir simultáneamente en el archivo, entonces la salida proveniente de las dos instancias de un componente, deben estar juntas en el archivo.

Se debe evitar el uso de variables globales temporales (en C), o de clases estáticas (en Java o C++), así como tener componentes stateful compartidos (que mantienen la información del estado de un recurso) y de recursos, como las conexiones a las bases de datos o los descriptores de archivos. En casos en donde los datos y los recursos se comparten, hay dos formas de fortalecer la seguridad de los hilos en un componente:

- Configurar el componente para ejecutarse con un solo hilo. Cada componente definido en el Jaguar Manager tiene una propiedad de hilos. Por default, los componentes son multi-hilos y las instancias pueden ejecutarse concurrentemente en diferentes hilos. También pueden tener un solo hilo. Cada invocación a un método bloquea a las otras, o a las instancias del mismo componente. En general, para los componentes, el manejo de un solo hilo es la peor alternativa, porque incrementa la probabilidad de que los clientes bloqueen a otras ejecuciones y que aumente el tiempo de respuesta para los usuarios finales. El manejo de un solo hilo tiene sentido para cierta clase de problemas, como compartir un archivo de salida entre las instancias de un componente.

- Almacenar datos compartidos en una base de datos remota. Se pueden usar cachés de conexión y almacenar los datos en una base de datos remota, dejando que el servidor de bases de datos maneje la concurrencia.

Los componentes que comparten los mismos recursos se deben codificar o configurar de forma que se evite un estado inconsistente.

III.5 Seguridad en Jaguar

Los riesgos que se corren al tener aplicaciones distribuidas, son los siguientes :

- Manipulación de la información que pasa entre el cliente y el host.

- Espionaje de las sesiones cliente.
- Hacerse pasar por un cliente o host diferente.

Existen mecanismos de seguridad, que se describen a continuación:

- **Criptografía de llave pública**, ofrece lo siguiente :
  - Autenticación: asegura que tanto el cliente como el servidor son quien dicen ser.
  - Encripción: modifica los datos de forma que puedan ser leídos sólo por la parte que se quiere que la lea.
  - Uso de un par de llaves para encriptar y desencriptar: llave privada-secreta y la llave pública, que es la más ampliamente distribuida.
  - Envío de la llave pública a cualquiera con quien se quiera comunicar usando datos codificados.
  - Los mensajes recibidos son encriptados con la llave pública distribuida y desencriptada con la llave privada.
  - Los mensajes enviados, son encriptados con la llave privada y desencriptados con la llave pública distribuida.

- **Secure Sockets Layer (SSL):** es un protocolo de red que brinda seguridad a las conexiones de red y usa la encripción de llave pública para proporcionar:
  - Autenticación de clientes y servidores usando certificados.
  - Encripción que previene que terceros entiendan los datos transmitidos.
  - Revisión de la integridad que detecta si los datos transmitidos han sido alterados.

Los paquetes para otros protocolos se pueden integrar dentro de los paquetes SSL. Una conexión en la que otro protocolo está integrado dentro del SSL es una conexión con tunelaje (Tunneling) SSL.

IIOP y HTTP pueden estar en forma de tunelaje dentro del SSL. Por ejemplo, el navegador Web crea una conexión HTTP segura cada vez que descarga una página de alguna dirección URL que empiece con HTTPS.

La seguridad es siempre una preocupación en las aplicaciones de tipo WebOLTP. Jaguar proporciona las siguientes ventajas en esta área:

- Soporte para el mecanismo estándar de seguridad en Internet SSL para encriptar los datos.
- Autenticación de usuarios a través de llaves públicas SSL.
- Seguridad a nivel aplicación con listas de control de acceso (ACLs).
- Programa de certificación *firewall.*

Los clientes pueden comunicarse con el servidor Jaguar, usando los protocolos seguros IIOPS y HTTPS, de la manera en que se ilustra en la siguiente figura.

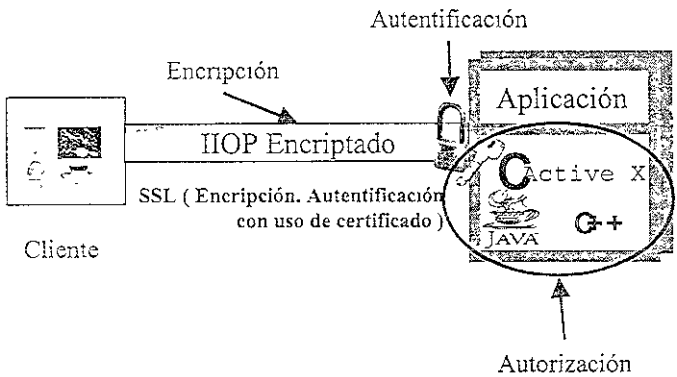


Fig. III.9 Características de seguridad

Se pueden utilizar las siguientes herramientas para administrar la seguridad en Jaguar:

- **Secure Manager :** administra los pares de llaves y certificados para Jaguar, y soporta un software confiable para activar el manejo de certificación y autenticación cliente/servidor.
- **Jaguar Manager :** establece perfiles de seguridad y los asigna a los oyentes (listeners).

En cuanto a la autenticación, la cual define quien puede establecer una sesión con el servidor Jaguar, se tienen las siguiente políticas :

- Autenticación del sistema operativo nativo.

- Autenticación personificada del código.
- Sin autenticación (default).
- Dirige las claves de acceso y passwords del sistema operativo al cliente Jaguar.
- El cliente Jaguar debe proporcionar un nombre de usuario y un password válidos en el servidor Jaguar.

En cuanto a la autorización, se pueden agregar perfiles a los paquetes y a los componentes Jaguar:

- Cada perfil puede incluir y excluir a usuarios y grupos específicos.
- Si el usuario no está autorizado, la creación del stub falla.
- En el Security Manager, se pueden agregar identificadores de certificados digitales a un perfil.
- La autorización para acceder a un componente, se basa en certificados SSL del cliente.

III.5.1 Encripción y autorización

Jaguar proporciona encripción y autorización por medio del SSL. Los navegadores pueden utilizar SSL para comunicarse con Jaguar. Además, Jaguar extiende el SSL a todas las comunicaciones entre Jaguar y el DBMS, en las cuales el SSL es soportado por los DBMS finales. El soporte SSL también se puede desconectar opcionalmente para mejorar la operación en redes internas seguras.

III.5.2 Seguridad a nivel aplicación

Jaguar limita la ejecución de los servlets a los usuarios autorizados por medio del uso de listas de control de acceso. El acceso puede ser limitado a paquetes, componentes e, inclusive, a un método específico en un componente. La seguridad es mantenida por el administrador.

III.5.3 Seguridad declarativa

Jaguar proporciona soporte para construir la autenticación y autorización del usuario. Los usuarios son autentificados cuando una aplicación cliente crea un objeto proxy o stub (una conexión se hace cuando la aplicación crea el primer objeto proxy o stub; otros proxies o stubs pueden usar las mismas conexiones o asignar nuevas conexiones según sea necesario). Cada componente tiene acceso a listas de control que

determinan a que usuarios se les permite invocar al componente; si un usuario no está autorizado para usar un componente, se crean stubs o proxies con error. Las opciones para validar al usuario son :

- Autenticación del sistema operativo nativo: Los nombres de usuarios para una conexión Jaguar se dirigen directamente a un nombre de acceso en el sistema operativo host.

- Autenticación codificada y personalizada: Se puede implementar e instalar un manejador personalizado para una solicitud de conexión del cliente

- Sin Autenticación: No se identifican los nombres de los usuarios de las sesiones. Ésta es la configuración por default de los servidores nuevos.

Jaguar cuenta con un nombre de usuario especial, *jagadmin*, que es la clave de acceso al Jaguar Manager. La identificación del administrador se realiza independientemente de la opción de autentificación que se configure. Por default, es el nombre de usuario *jagadmin* sin password.

El modelo de autorización de Jaguar se basa en perfiles, los cuales se definen en el Jaguar Manager. Cada perfil puede incluir y excluir nombres de usuarios específicos. Si se usa la autentificación del sistema operativo nativo, también se pueden incluir y excluir nombres de grupos de sistemas operativos. Todos los usuarios en el grupo específico son afectados.

Los perfiles se agregan a los paquetes y a los componentes de Jaguar, lo cual sirve para controlar el acceso a todos los componentes del paquete. Para usar un componente, se debe tener permiso de los perfiles del componente y del paquete que contiene al componente. El perfil de un paquete o de un componente controla el acceso de la manera siguiente:

- Los perfiles del componente son buscados en el orden en que aparecen en el Jaguar Manager.

- El primer perfil que específicamente excluya o incluya al usuario (o un grupo que contiene al usuario) determina si el acceso es otorgado o denegado. Si el usuario no está ni excluido ni incluido en algún perfil, el acceso es permitido.

"El progreso sólo se conseguirá abriendo los brazos al cambio continuo."

John Boorman.

Capítulo IV Jaguar CTS en el Web

IV.1 Conceptos Web

En este capítulo se verán los conceptos más utilizados para trabajar en red, ya que ésta ha cobrado gran importancia en el despliegue de aplicaciones remotas. Y como se ha visto, Jaguar CTS permite manejar transacciones en línea para muchos usuarios que acceden al mismo tiempo.

IV.1.1 World Wide Web (Red de redes)

La Internet y la Web no son lo mismo a pesar de que se utilizan como sinónimos. La Internet es una red global de interconexiones con otras redes, que proporciona el acceso a servidores, a información y a servicios a nivel mundial. Se basa en el protocolo TCP/IP. Las máquinas que acceden a Internet necesitan una dirección TCP/IP instalada y configurada, mientras que los servidores de Internet proporcionan diferentes servicios, como FTP, Telnet, Gopher o HTTP.

La World Wide Web, también conocido como Web, es una arquitectura computacional distribuida y es uno de los servicios que proporcionan los servidores de Internet. Puede estar desarrollado en Internet o en Intranet.

La Intranet es una red basada en la tecnología Web de una organización. Está desarrollada dentro de una red corporativa y en ocasiones proporciona acceso a Internet.

El Web se basa en una pequeña aplicación cliente que puede acceder a información que se encuentra en cualquier lugar mediante un navegador, a diferencia de las aplicaciones cliente/servidor que tienen clientes más grandes y especializados que interactúan directamente con una base de datos. La arquitectura Web, mostrada en la figura IV.1, está formada por las tecnologías que necesita, como son los protocolos TCP/IP y HTTP, y los navegadores y servidores Web.

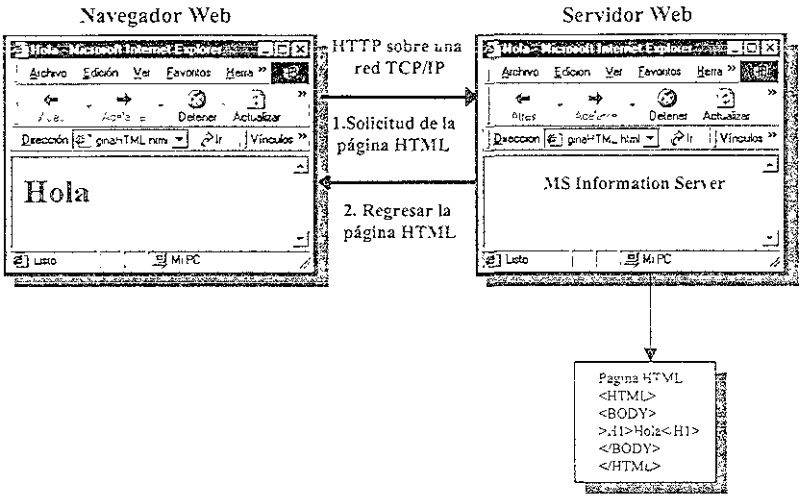


Fig. IV.1 Arquitectura Web básica

El servidor Web está dedicado a responder a solicitudes HTTP y devolver el documento adecuado. El navegador Web es una aplicación tipo cliente que puede comunicarse con el servidor Web a través del protocolo HTTP y solicitar, recibir, interpretar y desplegar la información enviada por éste.

Los navegadores Web se crearon para manejar varios tipos de archivos de aplicación por medio de la herramienta *built-in* y de los *plug-ins*. Algunos ejemplos de las clases de archivos que la mayoría de los navegadores pueden manejar, son los formatos de imágenes GIF y JPG, así como los archivos PDF de Adobe. Además, pueden ejecutar código mediante la interpretación de *scrips* insertados en documentos HTML, que generalmente están escritos en el lenguaje Java Script, y los principales navegadores tienen una máquina virtual de Java para ejecutar programas Java.

IV.1.2 HTML

La mayoría de la información que se encuentra en el Web, está almacenada en documentos HTML, los cuales son independientes de las plataformas y contienen información textual marcada con etiquetas (tags). Estos tags son usados para indicarle al navegador que haga algo especial además de desplegar texto. Se puede usar para definir la presentación de los datos y agregar información y recursos, como imágenes, archivos, scripts, controles y programas. También pueden proporcionar ligas a otros documentos que estén en el mismo servidor Web o en cualquier otro servidor Web accesible. A continuación se muestra un ejemplo de código HTML.

```
<html>
<body>
<H1>Hola</H1>
</body>
</html>
```

El navegador traduce las instrucciones para que el usuario vea lo que se muestra en la figura IV.1.

IV.1.3 URL

Cuando un navegador hace una solicitud al Web, lo hace usando el localizador de recursos uniforme (URL) estándar, el cual identifica el protocolo de comunicación, el archivo o el recurso que se está solicitando y en donde se localiza, incluyendo el servidor y la ruta. La sintaxis para el URL es la siguiente :

protocolo://host {:puerto}/path/nombre_archivo

IV.1.4 HTTP

HTTP significa Protocolo de Transferencia de HiperTexto. Es una aplicación a nivel de protocolo de comunicación que se ejecuta sobre TCP/IP. Permite que el contenido multimedia pueda ser accedido a través de la red para distintas plataformas. Es un protocolo *stateless* orientado a objetos que usa el modelo cliente/servidor para hacer solicitudes y darles respuesta. Una transacción o sesión HTTP está formada por cuatro pasos básicos:

- Conexión al servidor Web.
- Solicitud de un archivo o recurso.
- Recepción de un archivo, recurso o error.
- Desconexión del servidor Web.

La característica *stateless* del protocolo HTTP consiste en la desconexión del cliente del servidor una vez que éste ha respondido a la solicitud. Por ejemplo cuando se regresa una página HTML que contiene una imagen desde un servidor Web, ocurre lo siguiente: el navegador solicita una página HTML, para lo cual se conecta al servidor Web y hace la solicitud. El servidor Web maneja la solicitud, localiza el documento, envía la página HTML solicitada y se desconecta. El navegador traduce el documento HTML y presenta la información al usuario. Durante la traducción, el navegador se da cuenta de que se solicita una imagen debido a que encuentra el tag <IMG>.

Cuando una página HTML necesita recursos adicionales, en este caso, una imagen, debe hacer otra solicitud al servidor Web. El recurso adicional podría ser cualquier otro

tipo de contenido, como un applet. Para terminar de cargar la página HTML, el navegador se debe volver a conectar al servidor Web y solicitar la imagen. Después de que la imagen es recibida, se desconecta. Este proceso es repetido para todas las imágenes y recursos contenidos en el documento HTML, lo cual convierte a la arquitectura Web, inherentemente lenta.

La versión 1.1 del protocolo HTTP, permite hacer varias solicitudes en la misma conexión antes de desconectarse, lo cual acelera el regreso de los documentos HTML. Sin embargo, todavía no funciona como una conexión porque el navegador se desconecta del servidor Web después de que la página y todos los archivos que necesita son descargados.

IV.1.5 TCP/IP

Significa Protocolo de Control de Transmisión / Protocolo de Internet. Proporciona un conjunto de protocolos de comunicación de red que realizan las tareas de las capas de transporte y de red del modelo OSI (Interconexión de Sistemas Abierta). OSI fue desarrollado por ISO, y es usado para dividir los servicios proporcionados por los protocolos de red en siete capas. TCP/IP es uno de las pilas de red más comunes y es el protocolo estándar en Internet. Esto se debe a su amplio soporte de plataformas de hardware, como Windows, Unix, DOS y NetWare, y también a su amplio soporte de protocolos de red y de arquitecturas físicas, como Ethernet, Token Ring, Frame Relay y ATM.

IV.1.5.1 IP

El protocolo de Internet (IP) proporciona la comunicación de una red a otra, conocida como "ruteo". Para facilitar el ruteo, IP es responsable de proporcionar un esquema de direccionamiento para redes y de dirigir los datos a la máquina y red correctas. No se basa en la dirección física de cada nodo, sino que usa un esquema de direccionamiento lógico que permite a la red física cambiar sin tener que extender estos cambios a las capas superiores del modelo OSI. El direccionamiento lógico es convertido en un direccionamiento físico por medio del protocolo de resolución de direcciones (ARP).

Para identificar una máquina (o nodo) en una red, se necesita conocer la red en donde está conectada la máquina (identificador de red) y la dirección de la máquina en la red (identificador del host). La primer parte de la dirección es el identificador de red y define una dirección única en la red. La segunda, es el identificador del host que define una máquina única en la red. Sólo las máquinas que se encuentran en la misma red se pueden comunicar sin ruteador.

IV.1.5.2 TCP

Es el Protocolo de Control de la Transmisión. Establece una conexión y asegura que los mensajes de salida se dividan en paquetes para ser enviados dentro de una red

física, y que se vuelvan a armar en el orden correcto y sin errores para las siguientes capas.

Cuando el protocolo IP transmite datos, se encarga de llevarlos de cierta red a otra y de una máquina a otra. El protocolo TCP realiza el siguiente paso en este proceso, asegurándose de que los datos lleguen a la aplicación correcta una vez que han llegado a la máquina correcta. Para hacer ésto, usa un puerto que identifica la aplicación y el mensaje para ésta. Toda aplicación en un servidor debe tener un número de puerto único de 16 bits para identificarla. Hay varios puertos bien conocidos, como FTP (21), HTTP (80) y telnet (23).

Una máquina puede tener varias aplicaciones en un servidor, las cuales escuchan en diferentes puertos. Por ejemplo, un servidor de Windows NT puede tener el protocolo IIOP en el puerto 9000, HTTP en el 80, SMTP en el 25, FTP en el 21 y telnet en el 23. El protocolo IP asegura que la máquina correcta obtenga los datos que se están enviando, y el protocolo TCP de que el mensaje enviado no sea atendido por algún servicio distinto al que le corresponde. Cuando una conexión es establecida entre dos aplicaciones en una red TCP/IP, es llamada *socket*. Un *socket* es una conexión única formada por la combinación de una dirección IP y un número de puerto.

IV.1.5.3 UDP

TCP/IP actualmente tiene dos protocolos de transporte diferentes TCP y UDP. El segundo es el protocolo basado en *datagramas* para identificar a los usuarios, funciona sin conexión y se ejecuta sobre IP. Es más rápido que TCP ya que no proporciona una comunicación confiable entre hosts y no garantiza que los datos lleguen a su destino ni que estén libres de errores.

IV.1.6 Sitio Web dinámico

Un sitio Web básico, es en donde un navegador solicita un archivo HTML a un servidor Web, que recibe la solicitud de un documento HTTP, recupera el documento del sistema de archivos local y lo regresa al navegador. Estos documentos son páginas HTML estáticas, que son documentos fijos cuyo contenido no cambia a menos que el desarrollador edite el archivo y los vuelva a desplegar. Los sitios Web estáticos no muestran los datos en tiempo real ni permiten la interacción del usuario con el sitio.

El potencial de un sitio, está en ser dinámico, permitiendo la interacción con el usuario y el acceso a bases de datos. Un sitio dinámico permite que un servidor Web acepte una solicitud URL desde un navegador o de otro servidor Web, y regrese una página HTML dinámica generada en tiempo de ejecución, en lugar de regresar un documento estático desde un sistema de archivos. Una página Web dinámica es generada cada vez que se accede al sitio.

Debido a que las páginas dinámicas se generan al tiempo de ejecución, pueden responder a las entradas desde un navegador, regresar los datos solicitados por el

usuario, y acceder a la información en tiempo real. Una página HTML dinámica, comúnmente contiene scripts que se ejecutan en el servidor, que pueden acceder a un componente en un servidor de aplicaciones o a datos en una base de datos.

En general, los servidores Web no procesan la lógica del negocio ni acceden a las bases de datos. Esto no es parte de la funcionalidad requerida para recibir y procesar solicitudes de documentos HTTP. Sin embargo, debido al crecimiento de la Internet y del Web, los servidores Web cuentan con más características, como CGI, extensiones API (ISAPI, NSAPI), encripción de datos SSL y autentificación del servidor (certificados digitales) que permiten construir sitios Web dinámicos.

Para que un servidor Web regrese una página HTML dinámica, se necesita una interfaz entre el servidor Web y el programa o el servidor de aplicaciones que ejecuta el script. La interfaz del servidor Web estándar, es CGI (Interfaz de Gateway Común). Además de CGI, algunos proveedores de servidores Web ofrecen interfases propietarias para sus servidores, las dos más comunes son NSAPI (Interfaz de Programación para Aplicaciones de Netscape) e ISAPI (Interfaz de Programación para Aplicaciones de Servidores de Internet). ISAPI es la API que usan los servidores Web de Microsoft IIS (Servidor de Información de Internet). Además de ISAPI, Microsoft también tiene su propia máquina de ejecución de scripts llamada ASP (Páginas de Servidor Activas).

IV.1.7 CGI, NSAPI e ISAPI

CGI fue diseñado para permitir a los servidores Web acceder a programas externos para producir contenidos dinámicos. Estos programas externos pueden ser cualquier tipo de archivos ejecutables, como los archivos batch (por lote) de DOS, shells de Unix y los más comunes, programas PERL. Los programas externos pueden ser servidores de páginas dinámicas como Power Dynamo. CGI maneja todos los aspectos de la comunicación entre el servidor Web y otras aplicaciones, definiendo la forma en que los datos se transmiten. También describe las llamadas a las funciones. El resultado es regresada por el programa CGI al servidor Web y generalmente es un documento HTML, pero puede ser de cualquier tipo que el servidor Web maneje, como una imagen o una referencia a otro documento. Un programa CGI puede hacer casi todo, pero generalmente crea un documento HTML dinámico basado en las entradas del cliente y en el acceso a alguna fuente de datos.

CGI crea un proceso nuevo para cada solicitud, lo cual hace que el proceso sea lento e ineficiente. También usa variables de ambiente, y entradas y salidas estándares para recolectar y regresar información. La mayoría de los servidores Web tienen una interfaz propietaria que les permite ejecutar programas en el mismo espacio de procesamiento como si fuera él mismo, por lo que es más rápido y más eficiente. Las interfases propietarias más comunes son NSAPI e ISAPI.

Un programa CGI, NSAPI o ISAPI es llamado mediante un URL, parecido al llamado de un documento HTML estático. En lugar de especificar un documento HTML, se especifica un programa CGI.

Cuando el servidor Web recibe un URL que solicita una fuente CGI, llama a la aplicación externa y le pasa la solicitud a través de una interfaz CGI. El resultado del programa es regresado como un documento HTML que es enviado al servidor Web, el cual pasa este documento al navegador, como se muestra en la siguiente figura:

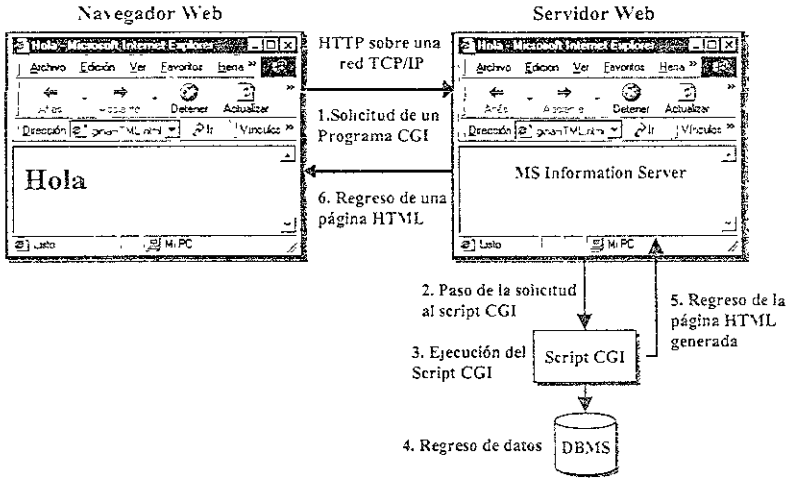


Fig. IV.2 Sitio Web dinámico que usa CGI

IV.1.8 Estructura de la Internet

La infraestructura básica de Internet es mostrada en la figura IV.3 y consiste de lo siguiente:

- Un navegador Web que presenta los datos a través de páginas HTML.
- Un servidor Web que almacena y maneja las páginas HTML.
- Comunicación estándar entre el navegador y el servidor mediante HTTP.

Esta estructura básica está diseñada para manejar información estática de aplicaciones publicitarias, como las promociones de mercado o la información de recursos humanos; y para tales fines es apropiada.

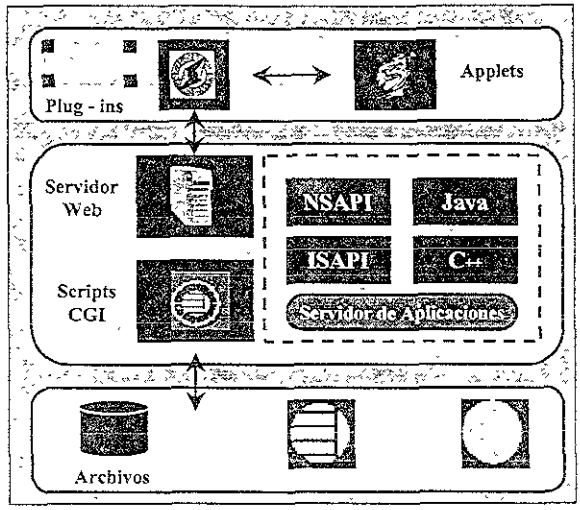


Fig. IV.3 Infraestructura básica de Internet con contenido dinámico

Como se muestra en la figura anterior, la infraestructura básica de Internet se ha extendido para permitir crear aplicaciones más dinámicas (usos interactivos), mediante la adición de:

- Formas simples basadas en JavaScript, y un navegador para recopilar y formatear los datos.
- APIs para servidores Web como NSAPI e ISAPI, que permiten que el servidor llame a las aplicaciones.
- Servidores de contenidos dinámicos, que transforman los datos de una base de datos en páginas HTML formateadas (ejem. PowerDynamo).

Al permitir manejar contenidos dinámicos, es decir, que los servidores puedan regresar los datos a un navegador para satisfacer las solicitudes del usuario o a otras formas de interacción, la infraestructura extendida del Web hace posible la creación de esta clase de aplicaciones, que van desde el soporte de decisiones a través de Intranet, hasta noticias personalizadas en Internet. Aún con estas extensiones, la infraestructura de Internet no puede manejar procesos transaccionales a gran escala. Hasta ahora, varias compañías han tratado de unir las bases de datos a los servidores Web. Pero sin las herramientas de desarrollo y la administración adecuadas, los resultados han sido deficientes en la mayoría de los casos.

IV.1.9 WebOLTP

El término WebOLTP describe a las aplicaciones que usan transacciones comerciales, ya sea en Internet, Intranet, Extranet, o en redes corporativas tradicionales. Sus características distintivas, en comparación con la OLTP que se basa en la arquitectura cliente/servidor, o con las tecnologías tradicionales de mainframes, son:

- **Clientes delgados:** En la arquitectura cliente/servidor tradicional, las aplicaciones requieren que se pre-instale el cliente, para poderse ejecutar. En la computación basada en clientes delgados, los componentes pueden ser descargados a través de un sitio Web. De esta forma, se obtiene un acceso universal y reducción de los costos de instalación y manejo. Los clientes delgados son parte inherente de la estructura básica de Internet. Sin embargo, requieren de tecnologías adicionales como Java para poder manejar aplicaciones dinámicas.

- **Alto volumen con muchas conexiones:** A diferencia de ciertas aplicaciones cliente/servidor y de sus predecesoras, las aplicaciones para la WebOLTP pueden extenderse debido al uso de Extranet o de Internet. Estas aplicaciones son accedidas por muchos usuarios que ejecutan transacciones simultáneamente, por lo que requieren de una arquitectura a gran escala.

- **Cargas de trabajo imprevisibles:** Las aplicaciones WebOLTP tienen cargas de trabajo imprevisibles. Cuando una aplicación Web está disponible, los administradores de los sitios Web difícilmente pueden predecir el número de usuarios y el momento en el que las descargarán, por lo que se necesitan sistemas altamente adaptables y flexibles para enfrentar cambios inesperados, y que además cuenten con tiempos de respuesta consistentes.

- **Aplicaciones de corta duración:** Cada generación de aplicaciones parece tener un tiempo de vida más corto que el de su predecesora. Conforme la tecnología de Internet se fortalece, las aplicaciones basadas en la red son desarrolladas en poco tiempo y están diseñadas para durar de 9 a 12 meses. Puesto que las expectativas del usuario común ha provocado que las aplicaciones WebOLTP sigan el mismo patrón, la tecnología para construirlas debe ser fácil de usar y desplegar.

La arquitectura de tres capas o multi-capas satisface las necesidades de la WebOLTP en cuanto a escalabilidad y acceso dinámico, al mismo tiempo que mantiene las características de la infraestructura Web básica. La arquitectura WebOLTP se muestra en la siguiente figura.

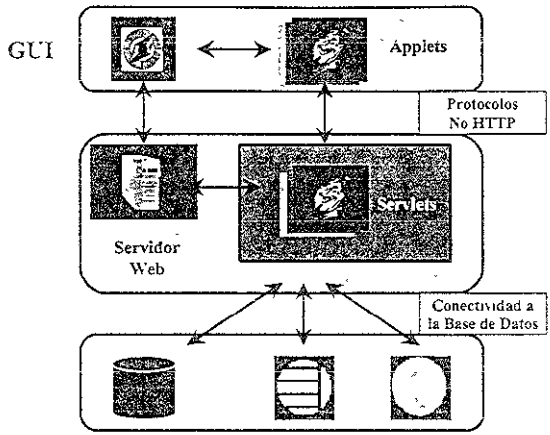


Fig. IV.4 Arquitectura WebOLTP

En este modelo, los usuarios encuentran y activan aplicaciones usando las páginas HTML tradicionales y los servidores Web. Pero además de descargar una página estática, descargan un applet dinámico en el navegador, junto con protocolos de alta velocidad, que le permiten comunicarse directamente con los servlets que se ejecutan en la capa intermedia, los cuales generalmente acceden a los datos de una o más bases de datos, aplican la lógica del negocio, y regresan los resultados al applet del cliente, en donde son mostrados.

Todavía existen dudas sobre las funciones exactas de cada componente y sus requerimientos técnicos clave para desarrollar sistemas escalables y fáciles de usar. El único protocolo que se encuentra comúnmente en uso extendido en la red, es HTTP, el cual fue parte de la infraestructura original de la WWW. Sin embargo, no funciona muy bien para transacciones interactivas de negocios, ya que carece de estado y de conexión, y está orientado a páginas.

Con el fin de superar estas desventajas, los expertos y vendedores han propuesto protocolos alternativos para la comunicación entre los navegadores y las capas intermedias. Las funciones requeridas para estos nuevos protocolos son:

- Mantener estados específicos de información de transacciones y usuarios.
- Flujo y manejo eficientes de los conjuntos de resultados.
- Soporte de transacciones.
- Ofrecer una encriptación segura de datos.

En una arquitectura multi-capas, las capas intermedias manejan la mayoría de los procedimientos de la aplicación. Es por esto, que la capa intermedia es una parte importante y controversial en la arquitectura WebOLTP.

Los DBMSs también son parte importante en todo sistema OLTP, incluyendo aquellos que están centrados en la WebOLTP. La optimización del DBMS de las aplicaciones WebOLTP, requiere de lo siguiente:

- Soporte de cargas de trabajo imprevistas, con características como hacer la solicitud para entrar a una lista de espera basada en prioridades.
- Conectividad a DBMS de alta velocidad desde Java.
- Aplicaciones de listas de espera y de control de recursos para reducir el uso de los recursos del sistema, y para predecir el funcionamiento de las transacciones basadas en Internet.
- Funciones de seguridad, como la autorización de un supervisor, la cual es necesaria para ciertas aplicaciones de la WebOLTP.
- Permitir incluir mayor variedad de tipos de dato.

IV.2 Acceso a componentes Jaguar desde el Web

Como ya se ha dicho, el servidor Jaguar es un servidor de aplicaciones que cumple con las especificaciones de CORBA y que ejecuta los métodos de los componentes con rapidez debido a su núcleo multi-hilos. Los clientes pueden acceder a los componentes por medio del protocolo Inter-ORB IIOP de CORBA, que permite que cualquier servidor de aplicaciones que cumpla con las especificaciones de CORBA 2.x se pueda comunicar con otros servidores y pueda acceder a otros componentes.

Para poder acceder desde el Web a los componentes que están en el servidor Jaguar, existen tres enfoques diferentes para diseñar una arquitectura Web que pueda ser implementada con Jaguar:

- Applet Jaguar – Java (Cliente Medio)
- Jaguar – Power Dynamo (Cliente Delgado)
- Servlet Jaguar – Java (Cliente Delgado)

El primer enfoque, Applet Jaguar–Java, permite que un navegador solicite páginas a

un servidor Web. Las páginas regresadas, contienen applets, los cuales también son descargados en el navegador por medio del protocolo HTTP. Después de haber sido descargado, el applet es el cliente primario usado para acceder al servidor Jaguar y llamar a los métodos de los componentes por medio del protocolo IIOP. Este enfoque se ilustra en la siguiente figura:

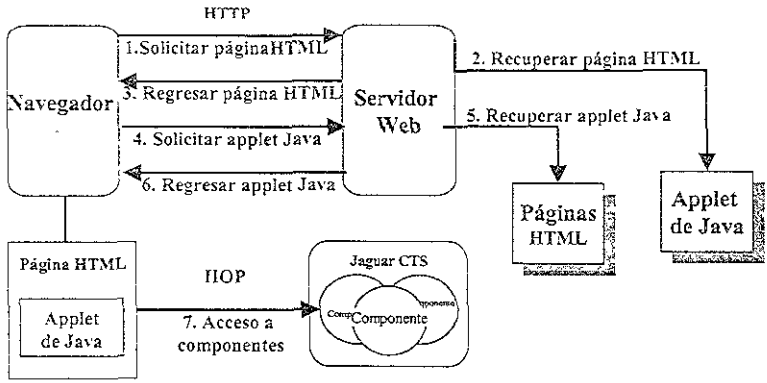


Fig. IV.5 Arquitectura del Applet Jaguar – Java

Este enfoque se puede implementar sin un servidor Web aislado, debido a que Jaguar soporta solicitudes HTTP y puede responder a un navegador con páginas estáticas y archivos.

Ventajas:

- Con los applets de Java se obtiene una GUI más robusta.
- Este método no necesita de un servidor de páginas dinámicas adicional.
- Hay una conexión continua y directa a Jaguar usando IIOP.

Desventajas:

- Los applets necesitan un navegador con Java habilitado y la versión correcta de la máquina virtual de Java.

- Los applets tardan más en descargarse que las páginas HTML.
- Las características del *sandbox* y de los *firewall* pueden hacer que la aplicación no sea una buena opción para usarse en Internet.

Una aplicación con un cliente ultra delgado, se puede construir combinando Jaguar con PowerDynamo, el cual es un servidor de páginas dinámicas basado en plantillas, que son archivos hechos por la combinación de HTML, tags de PowerDynamo y DynaScript. Este último es ejecutado por el servidor de PowerDynamo y los resultados se usan para construir una página Web dinámica al tiempo de ejecución. El script de la plantilla puede llamar a componentes Jaguar, accediéndolos como un cliente Java–CORBA. Los resultados de ejecutar los componentes se pueden usar en la página HTML dinámica que es generada y regresada al cliente. Este enfoque se ilustra en la siguiente figura:

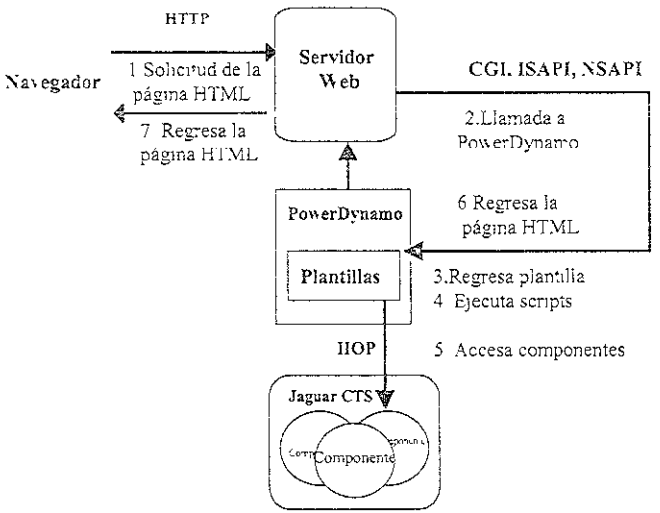


Fig. IV.6 Arquitectura Jaguar – Power Dynamo

Además de PowerDynamo, Jaguar puede ser accedido por cualquier otro servidor Web o de páginas dinámicas, que pueda acceder a objetos COM o CORBA, como ASP de Microsoft y ColdFusion de Allaire.

Ventajas:

- Requerimientos mínimos en las máquinas clientes (Cliente Delgado).
- Las páginas HTML son normalmente pequeñas y se descargan rápidamente.

Desventajas:

- Las solicitudes HTTP son *stateless* y necesitan que el servidor de páginas dinámicas o Jaguar manejen su estado.
- DynaScript es un lenguaje interpretado.

Un tercer enfoque es acceder desde un navegador a los servlets que están en el servidor Jaguar. Los servlets no son componentes Jaguar, son programas Java diseñados para ser llamados por solicitudes HTTP. Son similares a una plantilla de PowerDynamo en donde se puede acceder a componentes Jaguar y usar los resultados para construir una página Web o un archivo. Sin embargo, el servlet es cien por ciento Java, lo que hace que sea más fácil trabajar con éste que con DynaScript. Además son compilados en códigos de bytes, los cuales son más rápidos de interpretar que las plantillas Dynamo. Uno de los mayores inconvenientes de esta alternativa es que un servidor de aplicaciones Jaguar actúa como un servidor Web. Sim embargo, Jaguar no tiene la misma capacidad que éste. La arquitectura Jaguar–Servlet se ilustra en la siguiente figura.

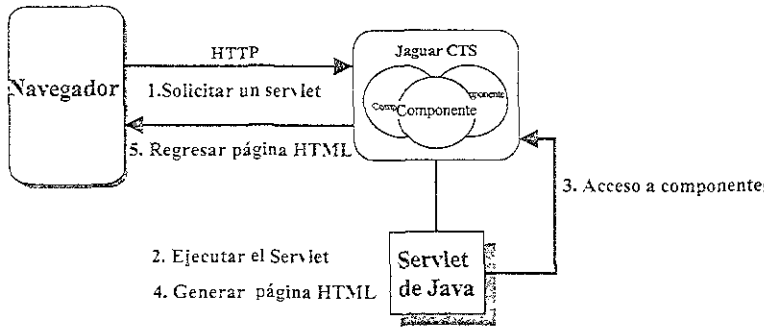


Fig. IV.7 Arquitectura Servlet Jaguar - Java

Ventajas:

- Se necesita poco o nada de software en las máquinas cliente.
- Permite que las clases de Java sean llamadas desde HTTP.
- El Java compilado es más rápido que las páginas interpretadas.
- Las páginas HTML son normalmente más pequeñas y se descargan rápidamente.
- Toda la codificación del lado del servidor esta hecha en Java.

Desventajas :

- Las solicitudes HTTP son *stateless* y requieren que el servlet maneje el estado.
- Utiliza a Jaguar como servidor Web para manejar las solicitudes HTTP.
- Necesita páginas HTML para ser construido con las clases Java, uniendo las cadenas HTML con datos.

Estos tres enfoques se pueden combinar para construir una aplicación Web dinámica. Existen otras técnicas para acceder al servidor Jaguar en el Web; pero las presentadas en este capítulo son las más sencillas y se basan en las prácticas estándares.

IV.3 Applets y Jaguar

Un applet es un programa pequeño del lado del cliente, escrito en Java y que es ejecutado en un navegador Web o en un visor de applets. No se puede ejecutar por sí mismo porque no es una aplicación de Java. Es usado para crear sitios Web dinámicos que extienden las capacidades de un navegador. Los applets son la respuesta de Sun a los controles ActiveX y a los *plug-ins* de Netscape del lado del cliente.

El navegador puede verse como un contenedor de applets que proporciona los procesos y los hilos en los cuales se ejecuta el applet. También controla el ciclo de vida del applet y le proporciona información a través de los métodos de su interfaz.

Existen dos métodos para la construcción de applets. El primero consiste en usar Java y JavaBeans. El segundo es la tecnología ActiveX. Sin importar que enfoque se use, los applets deben ofrecer los siguientes beneficios:

- Caracteres pequeños, para una descarga ágil.
- Una interfaz más interactiva y amigable que la de las páginas HTML comunes.
- Facilidad de desarrollo y mantenimiento.

Lo que hace interesante a los applets en una arquitectura Web, es que el programa del lado del cliente es descargado cuando se ejecuta. En una aplicación Web basada en *n* capas, a los applets también se les permite una conexión continua con los servidores de aplicaciones Jaguar, eliminando los problemas por la falta de estado del protocolo HTTP. Sin embargo, la solución es más que una aplicación Web híbrida, debido a que los applets y Jaguar se comunican usando el protocolo IIOP en lugar del HTTP.

Un applet está dentro de una página HTML y usa tags (etiquetas) HTML especiales. Esto es similar a insertar gráficas en una página HTML con el tag IMG. Cuando el navegador interpreta los tags, el recurso que se especifica es solicitado desde el servidor Web y es descargado. A continuación, el tag le dice al navegador que hacer con éste. En el caso de un applet, el archivo de clase es el recurso descargado. Cuando la clase es recibida, una JVM es iniciada en un hilo del proceso del navegador y la clase es instanciada. La arquitectura de un applet se ilustra a continuación:

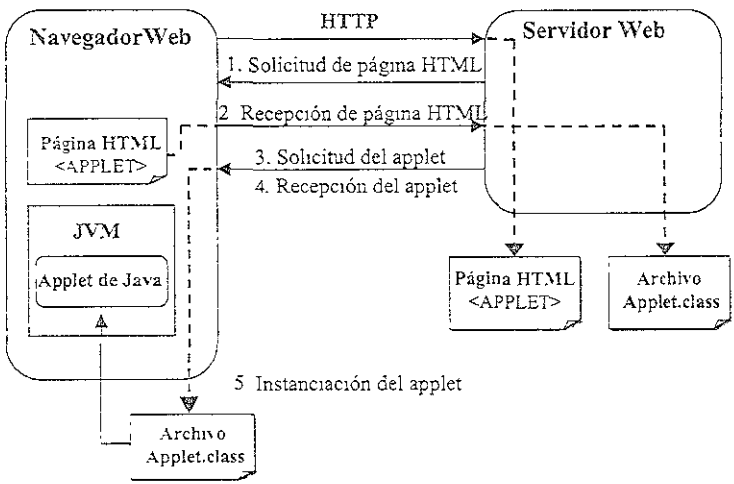


Fig. IV.8 Arquitectura de applets de Java

Cuando el applet se está ejecutando, se puede comunicar con el servidor de aplicaciones Jaguar como lo haría cualquier aplicación Java. Sin embargo, en la Internet los sitios Web frecuentemente son protegidos por los *firewalls*, los cuales pueden ser implementados con software o hardware. Una de las formas en que los *firewalls* evitan los ataques a los servidores, es bloqueando el tráfico que se dirige a los sitios Web que no usan HTTP. De esta forma, un applet no puede acceder al servidor Jaguar, ya que éste utiliza el protocolo IIOP. Lo anterior se ilustra en la siguiente figura.

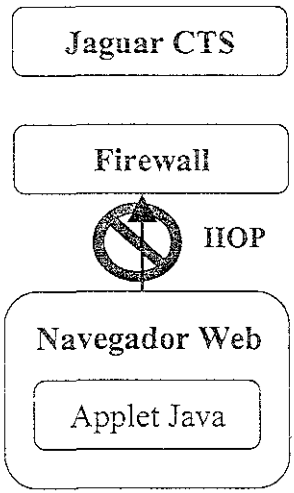


Fig. IV.9 La solicitud de applets no se puede obtener a través de un firewall

Los applets eran muy populares cuando se introdujo Java por primera vez. Sin embargo, tienen desventajas para las aplicaciones de Internet como, su lentitud para descargarse, los problemas con los *firewalls* y las incompatibilidades con las versiones de Java y de JVM. Los applets son mejores para las aplicaciones de Internet en donde el ambiente de las terminales del cliente se puede controlar y el ancho de banda es mayor.

IV.3.1 Los applets dentro de las páginas HTML

El tag APPLET es usado para incrustar un applet en una página HTML. Tiene un tag al inicio <APPLET> y un tag al final </APPLET>. Le indica al navegador que debe reservar espacio en el documento para el applet, según los atributos HEIGHT (alto) y WIDTH (ancho); y que clase de Java tiene que descargar, en dónde se localiza el

archivo de clases, y si se necesitan otros archivos. Los archivos y clases adicionales se guardan en un archivo JAR para facilitar su descarga y despliegue.

El atributo CODE del tag APPLET especifica el archivo de la clase Java que extiende a java.applet.Applet y que será cargado en la JVM del navegador. A continuación se presenta el ejemplo de un applet :

```
<HTML>
<TITLE>BANCO</TITLE>
<BODY>
<H1>BANCO applet</H1>
<HR>
<APPLET CODE = "BANCOapplet.class" CODEBASE = "classes/" ARCHIVE =
"BANCOapplet-JAR.jar" width = "361" height ="197" >
</APPLET>
<HR>
</BODY>
</HTML>
```

Se pueden tener tags <PARAM> entre los tags <APPLET> y </APPLET>, para especificar la información de inicialización en la página HTML para el applet. Esta información está definida en pares nombre/valor. El atributo NAME especifica el nombre que el applet puede usar para acceder al valor y VALUE define el valor asociado con un nombre. Un ejemplo del uso de los atributos de PARAM se muestra a continuación.

```
<APPLET CODE = "BANCOapplet.class" CODEBASE = "classes/" ARCHIVE =
"BANCOapplet-JAR.jar" width = "361" height ="197" >
<PARAM NAME = "PRUEBA" VALUE = "HOLA MUNDO">
</APPLET>
```

Para cambiar el servidor Jaguar al que accede un applet, se puede añadir el siguiente tag PARAM.

```
<APPLET>
<PARAM NAME ="JagURL" VALUE = "iiop://localhost:9000">
</APPLET>
```

Si el navegador está usando el *sandbox* para manejar applets, entonces el nombre del host de Jaguar especificado debe ser el mismo que el del host del servidor Web, sino,

la conexión a Jaguar será rechazada, debido a que los applets sólo pueden acceder al host en donde se originaron.

IV.3.2 Despliegue de un applet a Jaguar

Los archivos necesarios para desplegar un applet son:

- Clase del applet.
- Clases Java usadas por la clase applet.
- Archivo HTML con un tag APPLET para descargar el applet.

Jaguar no ejecuta applets, pero se puede usar para desplegarlos a los navegadores a través de una solicitud HTTP. La forma fácil de desplegar un applet formado por más de un archivo de clase Java, es juntarlos y ponerlos en un archivo JAR. El applet del ejemplo, usa muchas clases de Java adicionales para realizar sus tareas, incluyendo a las clases org.omg.CORBA y SessionManager para acceder a Jaguar, y el stubs Banco/Cuenta para obtener la referencia al componente Cuenta. El subdirectorio especificado en el atributo CODEBASE del tag APPLET debe estar disponible, o se debe incluir en el archivo JAR para que el applet funcione correctamente. Al desplegar el applet a Jaguar bajo el directorio %JAGUAR&\html\classes y al usar la sentencia CODEBASE = "classes/", se tiene el acceso a las clases adicionales al applet. Si el applet es desplegado a otro servidor Web o a un directorio diferente en Jaguar, entonces las clases referenciadas se deben incluir en el archivo JAR. Una vez que éste se ha construido, se necesitan desplegar los archivos HTML y JAR a Jaguar, el cual puede funcionar como servidor Web si cuenta con un listener (oyente) de HTTP activo. El listener HTTP por default tiene el nombre del host local y se ubica en el puerto 8080. Para aceptar solicitudes del Web desde clientes remotos, el nombre del host del listener HTTP debe cambiarse por la dirección IP o por el nombre de la computadora en donde está instalado Jaguar. Además, el puerto debe cambiarse a 80 para que Jaguar pueda aceptar solicitudes HTTP en el puerto estándar de HTTP.

El directorio raíz por default para las solicitudes HTTP en Jaguar es %JAGUAR%\html. Todos los documentos HTML deben colocarse en un subdirectorio bajo este directorio. El archivo JAR también puede estar en el directorio %JAGUAR%\html. Si se coloca en este directorio, no se requiere de un atributo CODEBASE. Sin embargo, el archivo JAR no necesita incluir a todas las clases referenciadas.

Es recomendable que cuando se instalen applets en un servidor Jaguar, las clases y/o los archivos JAR se coloquen en el directorio %JAGUAR%\html\classes. Esto permite que el applet encuentre cualquier clase de Java adicional que necesite. El atributo CODEBASE es necesario cuando la clase del applet y el documento HTML estén en directorios diferentes. Cuando el applet no puede encontrar las clases de Java que necesita localmente, intenta encontrarlas en el directorio especificado por CODEBASE. Si necesita clases adicionales que no están en el directorio %JAGUAR%\html\classes,

se pueden añadir para que sean accesibles cuando se les necesite. Una vez que el documento HTML y el archivo JAR son desplegados, y las clases adicionales están correctamente instaladas, el applet puede ser accedido a través del navegador.

IV.3.3 Sandbox

Una clase de Java no se ejecuta directamente en el CPU, sino dentro de la JVM, la cual interactúa con éste, interpreta el código de bytes al código máquina apropiado y proporciona un ambiente controlado para ejecutar la clase.

El navegador aprovecha ésto, implementando un administrador de seguridad que monitorea al applet y cuida de que no se desperdicien los recursos del sistema. Así mismo, refuerza las medidas de seguridad. Debido a que el applet es controlado por la JVM, y ésta es controlada por el navegador, estas medidas no siempre son reforzadas.

Debido a que el applet se ejecuta en un ambiente controlado, a éste frecuentemente se le llama *sandbox*. Los navegadores pueden restringir de diferentes maneras a un applet. A continuación se muestran las restricciones de los applets descargados:

- No puede ejecutar programas locales.
- No puede leer o escribir en archivos locales.
- No puede leer las propiedades del sistema.
- Puede hacer una conexión de red sólo con el host del cual fue descargado.

La restricción que hace rígido a un applet en Internet, es no poder acceder a una máquina diferente a aquella en donde fue descargada la clase de Java, como se muestra en la siguiente figura.

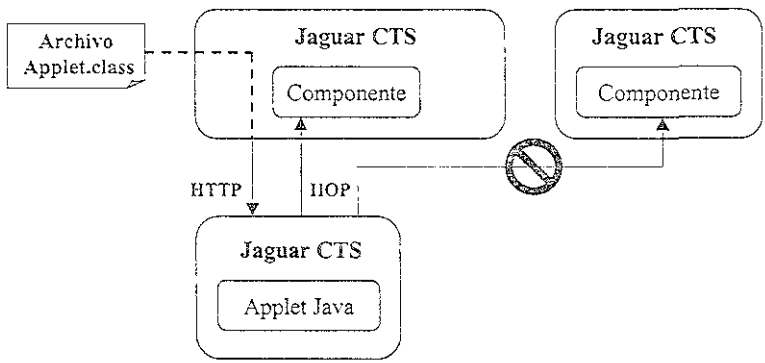


Fig. IV.10 Los applets de Java son restringidos por el sandbox

Esto hace difícil el balanceo de cargas, imposible el control de fallas, y la división de una aplicación en varios servidores Jaguar. El balanceo de cargas se puede implementar con clientes que usan diferentes servidores Jaguar en donde se descargan los applets, pero el *sandbox* no dejará que el applet acceda a otro host, para evitar fallas.

IV.4 Servlets y Jaguar

Los servlets son pequeños programas escritos en Java del lado del servidor que son usados para crear sitios Web dinámicos extendiendo a un servidor HTTP. Son la respuesta de Sun a los CGI y permite escribir programas en Java que responden a solicitudes HTTP. Se diseñaron para ser más rápidos que los CGI porque se ejecutan en el mismo proceso del servidor en lugar de empezar un nuevo para cada solicitud. La clase de Java que implementa al servlet, se coloca en memoria sólo una vez y permanece ahí después de usarse, lista para las siguientes solicitudes. También pueden manejar simultáneamente varias solicitudes mientras sean seguros de hilos (thread save).

Los servlets se usan para crear dinámicamente el contenido de un Web, como lo hacen los programas CGI, Powerdynamo y las ASP de Microsoft. Debido a que son programas en Java, también pueden usar la funcionalidad de la API de Java para procesar las solicitudes y generar un documento HTML. La figura IV.11 ilustra como un servlet básico responde a las socitudes HTTP desde un navegador. El servidor HTTP analiza el URL que solicita un servlet y pasa la solicitud al motor del servlet, el cual lo ejecuta y regresa al navegador un documento HTML dinámicamente generado. Un URL especifica un servlet usando un alias, que por default es /servlet seguido por el nombre del servlet, como se muestra a continuación:

http://localhost:8080/servlet/JavaServlet_Name

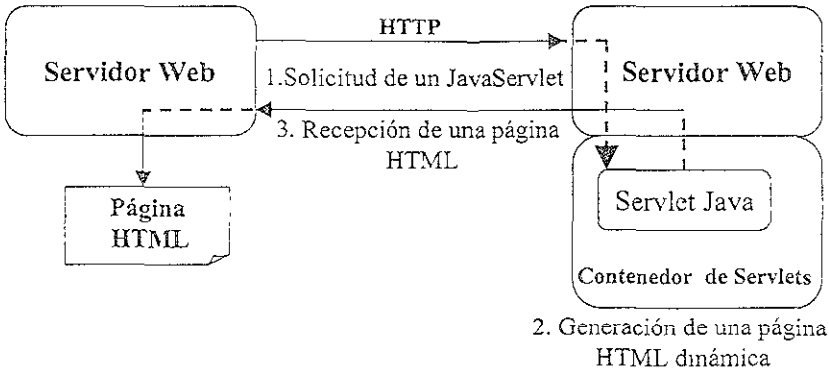


Fig. IV.11 Arquitectura básica de los servlets de Java

Los servlets, como todo programa Java, son independientes de la plataforma y del servidor. Se pueden ejecutar en cualquier servidor de aplicaciones que tenga un contenedor de servlets, el cual recibe la solicitud HTTP y la pasa a la instancia del servlet. El contenedor es un coordinador entre el servidor Web y la instancia del servlet, que administra desde la solicitud hasta la salida generada por el servlet, asegurándose de que sean manejadas correctamente. Para que un servidor HTTP ejecute servlets, necesita tener un motor y un contenedor que maneje las solicitudes y que añada la API y la especificación
del servlet.

Jaguar puede actuar como un servidor Web que maneja solicitudes HTTP, y que ejecuta servlets, los cuales pueden usar los cachés de conexión a bases de datos o llamar a los componentes para administrar el procesamiento de los datos y las solicitudes HTTP. Una página HTML puede presentar datos a través de una hiperliga o de un documento HTML. El servlet recibe la información por medio del navegador, llama a los componentes y pasa los datos como argumentos de funciones. Los componentes procesan la información e inicializan las transacciones devolviendo los resultados al servlet, el cual genera la respuesta HTML apropiada. La siguiente figura ilustra la arquitectura extendida de un servlet que usa Jaguar.

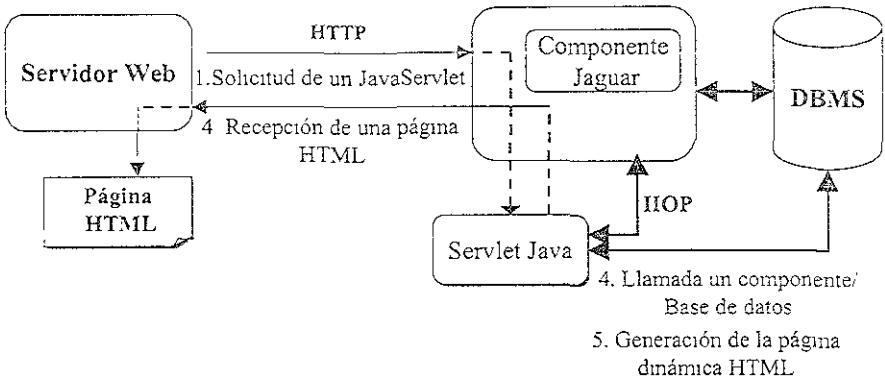


Fig. IV.12 Uso de los servlets para extender a Jaguar al Web

Los métodos que controlan el ciclo de vida de un servlet son: *init*, *destroy* y *service*.

- ***init***: es invocado cuando la clase Java es instanciada por primera vez. Después, la instancia atiende a las solicitudes HTTP que lleguen, y el servlet no puede ser accedido hasta que *init* haya terminado. Mediante este método se informa al servidor que el servlet se está ejecutando, se definen métodos que le permiten comunicarse con el contenedor, y la clase Java se inicializa apropiadamente.

- ***destroy:*** se invoca cuando un servidor descarga la clase Java que implementa al servlet, lo cual permite que la clase Java limpie cualquier recurso antes de que sea removida de memoria. Este método es llamado sólo una vez en la instancia de un servlet.

- ***service:*** es en donde el servlet maneja las respuestas a las solicitudes HTTP.

IV.4.1 Manejo de Servlets

Por default, sólo una instancia de la clase del servlet es colocada en memoria para manejar todas las solicitudes HTTP. Esta instancia recibe las llamadas al método *service* de varios hilos, los cuales representan a varios clientes. El servlet también debe poder manejar las solicitudes concurrentes que se están ejecutando al mismo tiempo en una sóla instancia. Para hacer ésto, debe contar con un manejo seguro de hilos y sincronizar el acceso a los recursos compartidos, como las variables de instancia, archivos, conexiones a bases de datos y referencias a objetos remotos. En la siguiente figura se ilustra como una sola instancia maneja simultáneamente varias solicitudes. La ejecución de cada hilo representa una solicitud HTTP diferente.

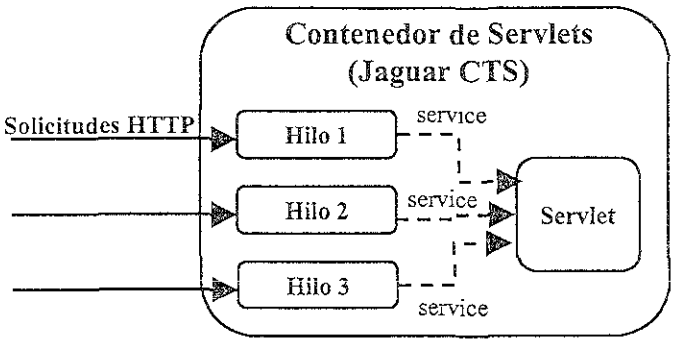


Fig. IV.13 El Servlet maneja varias solicitudes

Si la instancia de un servlet no se puede ejecutar con varios hilos al mismo tiempo, entonces debe permitir que el contenedor sepa que debe ejecutarse con un solo hilo. Las clases de Java pueden implementar la interfaz SingleThreadModel. Esta interfaz no añade ni métodos ni funcionalidad al servlet, sólo es utilizada para indicarle al servidor que la clase sólo usa un hilo. Cuando ésto pasa, sólo un hilo puede llamar al método *service* en una sola instancia del servlet en cierto momento. Las solicitudes adicionales son serializadas o encoladas hasta que el método *service* ha completado el procesamiento y la instancia está lista para recibir la siguiente solicitud. Esto se ilustra en la siguiente figura.

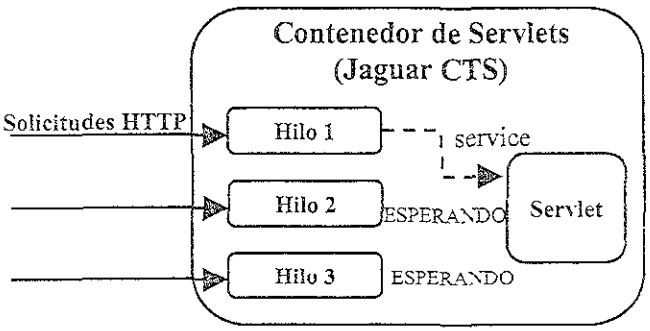


Fig. IV.14 Servlet que implementa la interfaz SingleThreadModel

V.4.2 Servlets y Componentes Jaguar

El funcionamiento de los hilos de un servlet instalado en Jaguar, puede configurarse en el Jaguar Manager. Sin embargo, cualquier clase del servlet que implemente la interfaz SingleThreadModel, se ejecuta en un solo hilo sin importar la configuración de las propiedades de los hilos de Jaguar.

Para configurar el funcionamiento de los hilos de un servlet instalado en Jaguar, se ilumina el servlet y se escoge la opción del menú File | Servlet Properties. Después la pestaña "Threading" y se habilita la opción "Single Threading" para indicarle a Jaguar *que realice en forma serial las llamadas para el servlet. Esto permite que el método service* ejecute sólo una solicitud en cierto momento dentro de una instancia del servlet.

Debido a que tratar de manejar muchas solicitudes serializadas con sólo una instancia del servlet no ayuda a mejorar el funcionamiento de una aplicación, Jaguar permite crear varias instancias de un mismo servlet y ejecutarlas en memoria, para mejorar el funcionamiento y eliminar el tiempo de espera para otros hilos. El campo "instancias máximas", permite configurar el número de instancias del servlet que Jaguar permitirá crear.

Cuando hay más de una instancia de un servlet ejecutándose, Jaguar dirige las solicitudes entrantes a la siguiente instancia disponible, pero si todas están ocupadas y ya se ha instanciado la cantidad máxima de clases, las solicitudes restantes tienen que *esperar hasta que el método service termine de ejecutarse en una de las instancias.* Sólo una solicitud sencilla se ejecuta en cualquiera de las instancias en cierto momento. Sin embargo, varias instancias de la misma clase pueden procesar las llamadas en el método *service* al mismo tiempo, como se muestra en la siguiente figura:

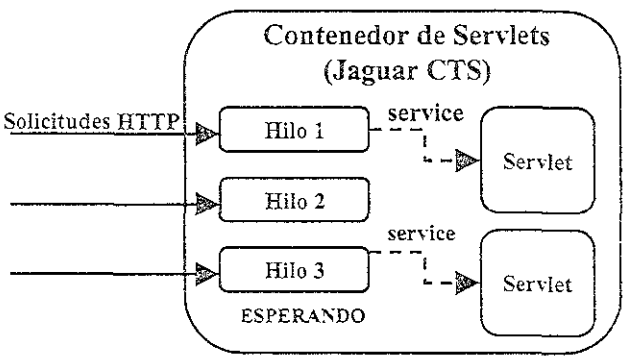


Fig. IV.15 Servlet de un solo hilo con múltiples instancias ejecutándose

Usar varias instancias en un solo hilo de una clase del servlet, garantiza que cada una tenga un manejo seguro de hilos, y que se proteja a las variables de instancia y a otros recursos internos. Sin embargo, no garantiza que el acceso a recursos externos sea un manejo seguro de hilos entre las instancias que se ejecutan al mismo tiempo.

Los servlets se ejecutan en el servidor Jaguar y pueden usarse para hacer aplicaciones Web dinámicas. A pesar de que pueden usar la API de Java para procesar la lógica del negocio y acceder a las bases de datos, no deben remplazar a los componentes Jaguar. Cada uno tienen sus propias funciones y juntos forman una arquitectura poderosa para aplicaciones Web. Sin embargo, los servlets sólo pueden ser llamados por clientes HTTP.

Debido a esas limitaciones, la lógica del negocio que se encuentra en un servlet, tiene sólo una solución Web. A pesar de que los servlets pueden usar cachés de conexión a bases de datos e invocar a los métodos de los componentes, no pueden participar en las transacciones de Jaguar. Por otro lado, un componente extiende la lógica del negocio a más clientes y devuelve objetos complejos y tipos de datos en su formato natural. Así mismo, participa en las transacciones manejadas por Jaguar ejecutadas por varios clientes diferentes (COM, CORBA, EJB), lo cual da mayor flexibilidad de uso y mejor implementación de la lógica del negocio y de las transacciones.

Los servlets son ideales para manejar cierto procesamiento de las páginas Web, ya que eliminan la lógica de presentación de los componentes de negocios en la capa intermedia. También se pueden usar para hacer que los componentes estén disponibles en el Web. Con el llamado de los componentes dentro de los servlets, éstos pueden ser accedidos por cualquier cliente.

La tabla siguiente resume las diferencias entre un servlet y un componente. Nótese que el componente no necesita escribirse en Java.

Tarea	Servlet	Componente
Implementa la lógica del negocio	si	si
Accede al caché de conexión a bases de datos de Jaguar	si	si
Accede a componentes Jaguar en el mismo o diferente servidor	si	si
Participa en una transacción Jaguar	no	si
Requiere un stub en el cliente	no	si
Solicitud HTTP disponible	si	no
Solicitud IIOP disponible	no	si

Tabla IV.1 Comparación entre los servlets de Java y los componentes de Jaguar

IV.4.3 Despliegue de los servlets a Jaguar

Una vez que la clase de un servlet se ha creado y compilado, los siguientes pasos son necesarios para desplegarla a Jaguar:

1. Copiar el archivo de la clase al servidor Jaguar.

2. Agregarla a "New Servlet" con Jaguar Manager.

3. Activar el servlet.

4. Probar el servlet con un navegador Web.

En el servidor Jaguar se crea un subdirectorio para la clase del servlet, bajo el directorio %JAGUAR%\java\classes, con el nombre del paquete. En el Jaguar Manager, se marca el servidor Jaguar, se selecciona el folder "Installed Servlets" y en el menú se escoge "Install Servlet", como se ve en la figura:

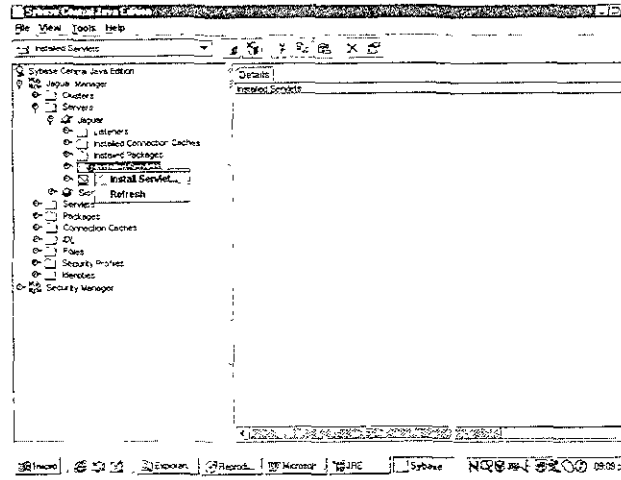


Fig. IV.16 Instalación de un servlet en Jaguar

El asistente para crear un nuevo servlet escribe su nombre, que es el del servlet que aparece en el Jaguar Manager y que será usado por la URL para acceder a la clase del servlet.

Cuando se marca la opción "During Startup" dentro de "Load", se indica a Jaguar que la instancia de la clase del servlet se debe instanciar cuando el servidor Jaguar se

activa, o hasta que se haga la primer solicitud al servlet. Debido a que el método *init* es llamado cuando se crea una instancia de la clase servlet, es recomendable haber hecho ésto cuando el servidor Jaguar se activa, evitando que la primer solicitud tenga que esperar a que la clase sea instanciada y a que el método *init* sea ejecutado.

Después de que el servlet es instalado por primera vez, se debe activar marcando el nombre del servlet que está en el folder "Installed Servlets" y seleccionando la opción "Start". El servlet necesita actualizarse cuando ya existe y se copia un archivo de clase con cambios al servidor Jaguar, o cuando se cambia alguna de sus propiedades. Al actualizarlo, se detiene y se vuelve a activar.

IV.4.4 Acceso de los servlets a los componentes

Un servlet puede ser usado para realizar muchos procesamientos de negocios complejos, ya que cuenta con la API de Java. Sin embargo, sólo puede ser accedido por las solicitudes HTTP, por lo que es mejor llamar a los métodos de un componente que implementen la lógica del negocio, que hacerlo desde un servlet.

Para que un servlet use un componente que cumpla con los estándares de CORBA, la clase Java debe tener acceso a los stubs Java de CORBA, generándolos desde el Jaguar Manager. Una vez generados y compilados (de tal forma que la clase del servlet tenga acceso a ellos), se puede planear como llamar a los métodos del componente. Hay tres técnicas que se pueden usar para obtener una referencia del objeto remoto al componente Jaguar:

- Referencia Interoperable (IOR).
- API CosNaming de CORBA.
- Interfaz SessionManager proporcionada por Jaguar.

La referencia interoperable (IOR) es la manera más fácil de acceder a un componente desde un servlet, ya que es parte del estándar de CORBA y define la sintaxis para referenciar un objeto como una cadena de texto codificada en hexadecimal que describe como conectarse al servidor al recibir el objeto. Sin embargo, para acceder a los componentes en el mismo servidor, el método string_to_object del ORB puede usarse con una cadena IOR definida en Package/Component. Usando este IOR se accede al componente en el mismo servidor Jaguar que el servlet.

El único inconveniente de usar el IOR, es que el componente debe existir en el mismo servidor que el servlet. Si no es así, se podrían usar los servicios de nombres de Jaguar a través de la API CosNaming de CORBA, o de la interfaz SessionManager.

IV.4.5 Propiedades de un servlet en Jaguar

En el servidor Jaguar existen varias propiedades para los servlets, las cuales afectan en la forma en que éste se ejecuta. Estas propiedades se ven en la siguiente figura:

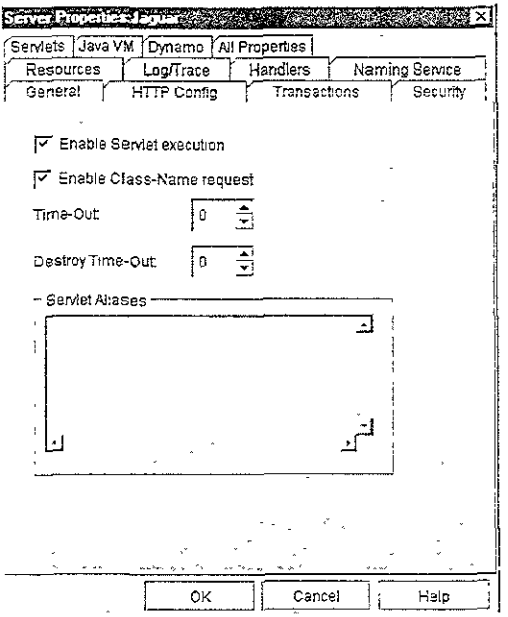


Fig. IV.17 Propiedades del servlet en el servidor Jaguar

A continuación se explican cada una de ellas:

- **"Enabled Servlet":** permite que los servlets se ejecuten en el servidor Jaguar. Por default está habilitada.

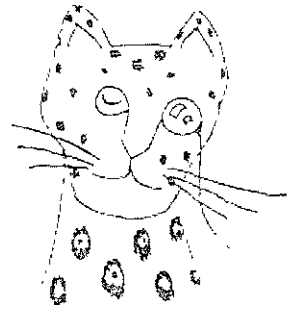
- **"Enabled Class_Name":** permite que se acceda al servlet por su nombre o por el de su clase.

- **"Time-Out":** le indica al servidor Jaguar cuanto tiempo en segundos tiene para ejecutarse el método *init* de un servlet. El default es cero, que le permite al método ejecutarse todo el tiempo necesario. Si esta opción es configurada y el método no termina de ejecutarse en el tiempo asignado, el servlet no estará disponible y se debe actualizar o reiniciar.

- **"Destroy Time-Out":** le indica al servidor Jaguar cuanto tiempo, en segundos, debe esperar después de que finalice o se actualice el servlet. El default es cero, que permite llamar inmediatamente al método *destroy.*

"Cualquier cosa que hagas no está terminada hasta que verifiques que hiciste lo que realmente querías hacer."

Ivar Jacobson



Capítulo V Aplicaciones

V.1 Diseño y construcción de aplicaciones Jaguar

Las aplicaciones Jaguar están formadas por un cliente que se encuentra en una máquina y varios componentes que están en otra, éstos últimos se pueden conectar a las bases de datos de otras máquinas. La construcción de aplicaciones Jaguar es diferente a la de aplicaciones estándares, ya que sus partes se comunican entre sí a través de la red bajo una arquitectura de tres capas. Los clientes residen en la primer capa, el servidor y los componentes en la segunda, y las bases de datos en la tercera. Debido a que Jaguar maneja los detalles de las transacciones, los hilos, la seguridad, las conexiones a bases de datos, y la comunicación de red, el desarrollador se puede concentrar solamente en escribir la lógica del negocio.

Como en las aplicaciones tradicionales cliente/servidor, el cliente de esta arquitectura contiene la interfaz de usuario, pero la lógica del negocio está separada del servidor y reside en la capa intermedia dentro de componentes que analizan datos, realizan cálculos, o regresan información desde fuentes de datos y la procesan. El diseño de una aplicación Jaguar se hace codificando estas tareas en un prototipo de la interfaz y de los métodos.

Una aplicación puede tener componentes preconstruidos o se pueden importar mediante el Jaguar Manager. Al importarlos, se añaden a Jaguar los prototipos de sus interfases y de sus métodos.

El cliente y los componentes se pueden construir paralelamente, y los desarrolladores se pueden notificar cuando hagan algún cambio en el prototipo de la interfaz o de los métodos.

V.1.1 Construcción del cliente

Primero se decide si es Java o ActiveX el que mejor se ajusta a las necesidades. Los applets no necesitan instalación personalizada y simplifican la tarea de las actualizaciones. El cliente siempre descarga el applet más reciente, y para que no espere a que las clases de Java se descarguen desde el servidor Jaguar, se pueden instalar en la máquina cliente. Si la aplicación cliente es grande y necesita muchas clases, el tiempo de descarga podría ser inaceptable. En este caso, se usa una aplicación Java instalada en la máquina cliente, la cual es ideal para los clientes de Intranet o de Internet. A pesar de que no son tan sencillas de usar como un applet, no son más difíciles de actualizar que el software convencional. Jaguar no restringe a desarrollar solamente clientes Java, también se pueden hacer ActiveX (como Visual Basic), los cuales requieren de una instalación local. Al diseñar el cliente, se debe

planear la optimización del funcionamiento de la red, procurando que sea mínimo el tráfico entre el cliente y los componentes que se encuentran en el servidor. Para optimizar el funcionamiento de la red, se planea lo siguiente:

- Cambiar la propiedad "Caché" en las estructuras de datos del cliente.
- Validar los valores de los campos en el cliente.
- Actualizar sólo los renglones y las columnas que hayan cambiado. Por ejemplo, no se permite a un cliente Java actualizar una tabla completa cuando sólo han cambiado algunos renglones.
- Agrupar los cambios de los datos para hacer menos llamadas a los métodos.

Para construir clientes Java se hace lo siguiente:

- Generar stubs de Java para que los componentes ejecuten sus métodos.
- Crear un applet de Java o una aplicación usando el JDK 1.1 o 1.0.2 de Javasoft. También se pueden usar las herramientas de desarrollo de Java, como PowerJ o Symantec Cafe, basados en el JDK 1.1 o 1.0.2 de Javasoft.
- Crear una página HTML para descargar el applet y sus clases, o instalar la aplicación Java en el cliente.

Para construir clientes ActiveX se hace lo siguiente:

- Exportar librerías de tipo y registrar los archivos para los componentes en los cuales el cliente ejecutará los métodos.
- Usar un IDE con ActiveX habilitado.
- Instalar la aplicación en el cliente.

V.1.2 Construcción de Componentes

A continuación se describe la forma en que se construyen los componentes de distintos tipos.

- **Componentes Java:** se usa el JDK 1.1, posteriormente se importa la definición del componente dentro de Jaguar, y finalmente se instala en el servidor Jaguar.
- **Componentes ActiveX**: se usa un IDE, se importan las definiciones ActiveX para el componente dentro de Jaguar, y se instala el componente en el servidor Jaguar.

- **Componentes C:** se generan los skeletons desde el prototipo del método del componente definido en el Jaguar Manager, se codifica el cuerpo del método en las plantillas de la implementación del método, y se compila e instala la DLL de C en el servidor Jaguar.

V.1.3 Construcción de aplicaciones Jaguar

Una aplicación Jaguar consiste de uno o más paquetes y un cliente o applet. Los paquetes consisten de componentes, los cuales están hechos de uno o más métodos. Un paquete es una colección de componentes que trabajan en conjunto para proporcionar un servicio o implementar la lógica del negocio. Jaguar soporta los siguientes tipos de componentes:

- ActiveX (Visual Basic, Visual C, etc.)
- C
- CORBA C++
- Java
- PowerBuilder
- DLL o C (Delphi, FoxPro, etc.)

Hay tres pasos principales para crear una aplicación en Jaguar:

1. Definir paquetes, componentes y métodos. Para definir un paquete se hace lo siguiente:
   - Crear un nuevo paquete
   - Instalar el paquete en algún servidor.
   - Modificar las especificaciones del paquete.
   - Configurar las propiedades del paquete.
2. Crear applets y componentes. Primero se crean los componentes que se instalarán en el EAServer, y posteriormente se generan los stubs y skeletons con Jaguar Manager. Después se decide el tipo de cliente que se va a desarrollar. Se puede hacer que más de un cliente interactue con los componentes creados. De esta forma, los diferentes usuarios pueden acceder a la aplicación desde otros medios, como un navegador, un cliente en PowerBuilder, etc.
3. Liberar la aplicación. Se pueden registrar componentes en cualquier servidor Jaguar. Una vez probada y revisada la aplicación, se deben pasar los componentes del servidor de desarrollo al servidor de producción. Para ésto, Jaguar tiene la opción de exportación e importación de archivos.

A continuación se muestra la secuencia de estos pasos en la figura.

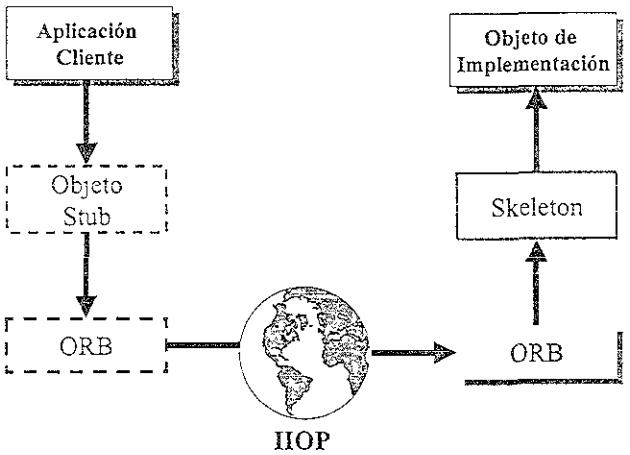


Fig. V.1 Cliente y Componente en Jaguar

La liberación del cliente depende del lenguaje con que fue creado. Se debe consultar la información del proveedor correspondiente para cada tipo de cliente.

V.2 Ejemplos de Aplicaciones

A continuación se muestran algunos ejemplos de requerimientos para los cuales una solución podría ser una aplicación que trabaje con una arquitectura de tres capas y que tenga a Jaguar CTS como servidor de componentes en la capa intermedia.

Distribución de botanas	Consulta de los precios e inventarios a través de Internet. Emisión de facturas en cuanto el producto sea enviado. Terminales móviles para acceder a los sistemas de inventarios y a las listas de precios. La negociación de precios se haría en el mismo lugar de la venta.
Cadena de librerías	Búsqueda de inventarios y órdenes de pedidos a través de Internet. Permitir buscar en todo el inventario de la cadena, localizar el libro, marcarlo como vendido, proporcionar las instrucciones del envío al lugar en donde se localizó el libro y cerrar la venta.
Inscripciones Escolares	Inscripción de estudiantes, consulta de calificaciones e historial académico a través de Internet. Selección de las materias a las que podrían inscribirse y al mismo tiempo confirmación de la inscripción realizada. Consulta de la lista de libros para cada curso; con opción de compra y de entrega personal o a domicilio. Se podrían hacer los pagos mediante transferencias bancarias a la cuenta de la Universidad.
Servicios Financieros	Envío de una orden a un cliente, quien la llenaría y regresaría por Internet. Al mismo tiempo la información solicitada aparecería automáticamente en la pantalla del vendedor, de forma que esté atento de lo que está viendo el cliente. Estos avances eliminarían los retrasos que ocurrirían si se usara el fax y el correo electrónico .

V.3 Aplicación Ilustrativa

En este apartado se describirán algunas aplicaciones que utilizan la tecnología de Jaguar. Este ejemplo es buen punto de referencia para evaluar el uso que ha tenido esta tecnología, y para analizar de forma práctica el funcionamiento de esta arquitectura.

La aplicación desarrollada es un sistema distribuido en la Intranet de la compañía **Ericsson**, la cual accede a la información y la procesa, mediante el uso de una arquitectura multi-capa con componentes y Jaguar CTS en la capa intermedia. La compañía consultora que desarrolló el sistema fue Upright Engineering.

Ericsson es la compañía líder de la industria de las telecomunicaciones que ofrece

soluciones avanzadas de comunicación para redes fijas y móviles, así como productos de consumo. Proporciona soluciones totales para todo tipo de clientes, como los operadores de redes y de servicios, las empresas y los consumidores. Tiene más de 100,000 empleados y representación en 140 países, además de la base de clientes más grande del mundo en su campo. Su oficina matriz se encuentra en Suecia, y su fuerza radica en su presencia internacional, su investigación, su desarrollo a nivel mundial, su variedad de soluciones fuertes para todo tipo de clientes, y su liderazgo en tecnología wireless y móvil.

- ***Antecedentes***

Ericsson mantiene bases de datos distribuidas que contienen la información de las modificaciones de los códigos fuentes de los módulos del software que residen en estaciones AXE. Se buscaba una herramienta que mejorara el proceso de actualización y distribución de las versiones nuevas del software, lo cual implicaba comparar, actualizar y crear nuevos documentos basados en la información existente en las bases de datos y en la necesidad de incluir la información de documentos complementarios de nuevas fuentes.

Ericsson definió los requerimientos funcionales para la herramienta:

- La aplicación debería ser ejecutable en la Intranet.
- La interfaz de usuario debería estar basada en un navegador Web.
- La aplicación debería funcionar en diferentes sistemas operativos.
- La aplicación debería funcionar para muchos accesos simultáneos.
- La aplicación debería ser parte de un sistema más grande. Lo que implicaba que la lógica del negocio se implementara en componentes para soportar la integración seamless con otras aplicaciones.
- La aplicación debería ser escalable e implementada con una arquitectura moderna que asegurara una solución futura.

Para satisfacer los requerimientos, Upright Engineering diseño una arquitectura de tres capas basada en componentes. AEServer de Sybase para la base de datos, Jaguar CTS para la capa intermedia, y la capa cliente sería implementada como un cliente Web delgado. Las pruebas demostraron que la solución era escalable, rápida, confiable y completamente capaz de interactuar con otros sistemas.

- ***Planteamiento del problema***

La información relacionada con los cambios y las modificaciones de los módulos de software que residen en las estaciones AXE de Ericsson, son almacenadas en diferentes tipos de documentos. En un proceso de actualización y modificación, la información de estos documentos se debe comparar, modificar, y a veces mezclar con información adicional de alguna base de datos. Los estudios realizados revelaron que había un considerable ahorro de tiempo si los procesos de actualización de software se automatizaran y fueran soportados por un conjunto de herramientas dedicadas.

La siguiente figura ilustra el conjunto de documentos basado en la información de varias fuentes de datos.

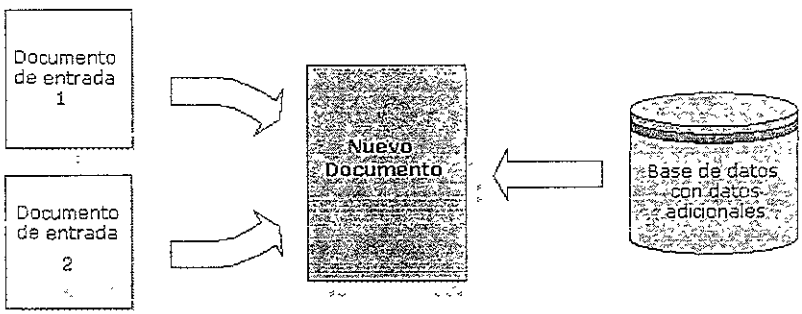


Fig. V.2 Ensamble de documentos

Uno de los objetivos era contar con equipos de desarrollo distribuidos globalmente que trabajaran en ambientes UNIX y NT, por lo cual, la herramienta debería ser accesible desde la Intranet de la compañía mediante un navegador Web, y debido a ésto, no era importante conocer el número de usuarios que accederían simultáneamente.

La herramienta sería una parte integrada a un paquete completo de desarrollo y mantenimiento. Esto implicaba que los procesos de negocios deberían diseñarse como parte de un todo y no como aplicaciones aisladas. Debido a que la herramienta jugaría un papel clave en mejoras futuras del ambiente de desarrollo, los requerimientos ideales se consideraron como pruebas futuras de diseño y solución.

Junto con el equipo de desarrollo del cliente, Upright Engineering analizó el problema. El resultado se puede ver como una evolución de los requerimientos a la visualización de la necesidad de una arquitectura multi-capa, con temas como la integración *seamless*, la escalabilidad, las aplicaciones basadas en componentes para lograr

independencia de plataformas y los clientes delgados, así como a la integración de la información existente.

- ***Arquitectura del Sistema***

El sistema es una arquitectura de tres capas, basada en componentes con clientes delgados. Se escogió AEServer como el servidor de bases de datos y a Jaguar CTS de Sybase como el servidor de la capa intermedia. La siguiente figura presenta un esquema global del sistema.

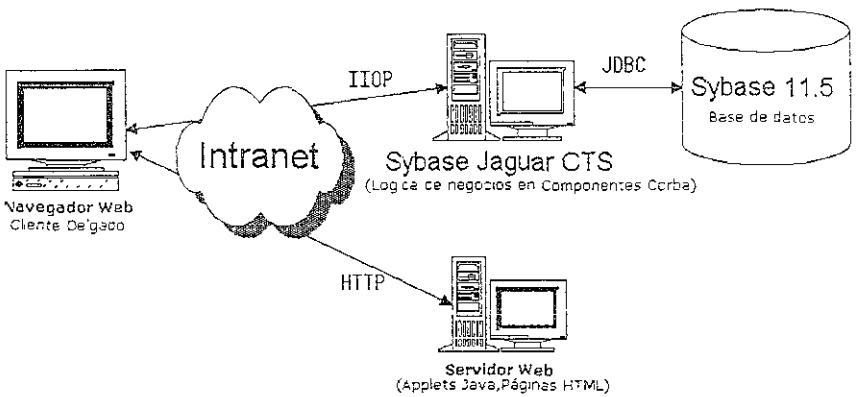


Fig. V.3 Esquema de la aplicación

La interfaz gráfica de usuario (GUI) se localiza en el servidor Web, implementado como un applet y las aplicaciones de procesos de negocios están en el servidor Jaguar implementadas como componentes Java.

Para iniciar el trabajo en el sistema, el usuario debe introducir la dirección para llamar a la función desde el navegador. Las páginas HTML relacionadas y los applets son descargados desde el servidor Web usando la comunicación tradicional HTTP. El applet principal contacta al servidor Jaguar, que regresa la referencia al componente correspondiente. El protocolo de comunicación entre el cliente y los componentes es CORBA/IIOP. Todos los procesos lógicos son implementados como componentes escritos en Java, que residen en el servidor Jaguar.

Ninguna llamada a la base de datos se realiza directamente desde el cliente, sino a través de los componentes. Este enfoque mejora el sistema en términos de seguridad y funcionamiento. La comunicación entre la base de datos y el servidor Jaguar es

manejada por una conexión JDBC.

Para este proyecto, Upright Engineering escogió implementar componentes escritos en Java, que se comunican mediante CORBA/IIOP. Esta opción permite escoger, ya que los componentes se pueden escribir en C y ser llamados por clientes Java y de forma contraria, una aplicación en C++ puede llamar a componentes implementados en Java. Además, los componentes pueden llamar a otros componentes.

La decisión de usar Jaguar fue tomada por el equipo de desarrollo del cliente, que se basó en el amplio conjunto de funcionalidades que ofrece este producto, y especialmente en las diferentes opciones de componentes y de clientes con las que cuenta. Además de su compatibilidad con CORBA.

- ***Implementación de la lógica del negocio***

Para asegurar una implementación eficiente de los procesos, Upright realizó un estudio profundo de la forma en que los documentos fueron estructurados. Los distintos tipos de documentos se asignaron de acuerdo a un criterio de *parseo* individual, el cual es un medio para identificar y extraer la información relevante y relativa al tema. El criterio de parseo fue definido a un nivel en donde podrían ser usados como la base para escribir el código para las funciones. Se creó un componente para cada tipo de documento. Los componentes contienen las reglas específicas para cada tipo de *documento y los métodos para hacer el parseo y modificar los documentos.*

Hubo muchas variantes de comparación. Sin embargo, ya que estas funciones trabajan de manera similar, el enfoque lógico fue implementarlos en un solo componente que tuviera los métodos para introducir los datos de la base de datos en los documentos.

Una vez que los procesos del negocio se implementan como componentes, cada vez que hay alguna modificación o mejora en el componente, sólo se compila y se actualiza el servidor para que se efectúe el cambio. La próxima vez que se llama al componente, las funciones actualizadas toman efecto sin necesidad de actualizar las aplicaciones cliente.

La interfaz de usuario fue diseñada trabajando de cerca con los usuarios. Esto ayudó a que los desarrolladores entendieran como se quería utilizar el sistema, y los usuarios finales obtuvieron un ambiente de trabajo que encontraron fácil de usar y que iba de acuerdo a sus procesos de trabajo.

- ***Aspecto escalable***

Una preocupación mayor, fue la necesidad de escalar el sistema cuando las condiciones cambiaran. Esto fue especialmente importante, ya que la aplicación iba a ser usada por un número desconocido de usuarios, que se incrementaría sobre la Intranet. La escalabilidad se logra usando componentes *stateless*, el administrador de

transacciones de Jaguar y los cachés de conexión.

- ***Servidor Web***

La Intranet generalmente tiene varios servidores Web activos. Es ideal que los archivos del cliente, los applets y los archivos HTML residan en esos servidores. La arquitectura escogida soporta este requerimiento. El applet y sus archivos HTML correspondientes fueron copiados a los servidores Web respectivos, y el administrador se responsabilizó e integró el trabajo hecho por Upright con la estructura Web existente.

Los servidores Web no tienen un papel importante en el sistema, sólo son en donde se empieza a trabajar. Una vez que el navegador ha descargado los archivos necesarios para ejecutar la aplicación, ya no hay más operaciones entre el cliente y éste. El servidor Web incluido en Jaguar es suficiente en la fase de desarrollo, pero cuando el sistema está en operación, es recomendable usar un servidor dedicado exclusivamente a este fin.

La interfaz de usuario se implementó con applets de Java, para lo cual se usó PowerJ. Para que un applet utilice un componente, sólo necesita obtener su referencia, que está en el servidor Jaguar, y almacenarla en un objeto. Las llamadas al método del objeto son iguales a las de cualquier objeto de otra clase. Para programar las aplicaciones cliente, sólo se necesita conocer la interfaz del componente y la forma de obtener su referencia, sin importar que lenguaje se utilice. Sólo se requiere hacer cambios en esa interfaz. Sin embago, ésto se considera una modificación mayor que no se debe hacer innecesariamente.

- ***Cliente***

La interfaz de usuario se implementó con applets de Java, para lo cual se usó PowerJ. Para que un applet utilice un componente, sólo necesita obtener su referencia, que está en el servidor Jaguar y almacenarla en un objeto. Las invocaciones al método del objeto son iguales a las de cualquier objeto de otra clase. Para programar las aplicaciones cliente, sólo se necesita conocer la interfaz del componente y la forma de obtener su referencia, sin importar el lenguaje. Además de los cambios en la definición de la interfaz del componente, no se requiere de otras modificaciones en la aplicación cliente. Alterar la interfaz del componente se considera una modificación mayor y no se debe hacer innecesariamente.

- ***Servidor de Bases de Datos***

La base de datos seleccionada fue Adaptive Enterprise 11.5 de Sybase. El equipo de desarrollo escogió continuar con una estructura de datos simple. La base de datos sólo contiene datos y no la lógica del negocio.

- ***Ventajas del Sistema***

Además de proporcionar estabilidad y velocidad, el sistema es escalable y puede atender a muchos usuarios simultáneamente. En el sistema, los componentes se comunican usando CORBA/IIOP, lo cual les permite interactuar con otros sistemas, por ejemplo con sistemas propietarios.

La introducción de una nueva arquitectura no implica reemplazar a los sistemas existentes. El uso de la tecnología de componentes permite la integración con estos sistemas. Por ejemplo, las nuevas funciones que usan las bases de datos existentes, se pueden implementar usando el mismo enfoque de componentes que utilizó Upright Engineering en el proyecto.

El mantenimiento del sistema ha sido fácil, debido a que no hay instalaciones en el cliente. Además, el uso de una arquitectura basada en componentes reduce el riesgo de que el sistema sea obsoleto en el futuro.

V.4 Aplicaciones Actuales

En este apartado se describen las experiencias prácticas y reales que actualmente han tenido diferentes tipos de compañías a nivel internacional, al utilizar la tecnología de Jaguar y de Java en arquitecturas distribuidas de tres capas.

V.4.1 Ericsson

De nueva cuenta, se presenta una aplicación desarrollada por Ericsson. Este caso se trata de la preparación y distribución de reportes de resultados de las oficinas internacionales de Ericsson para los administradores.

Con operaciones en todo el mundo, un requerimiento importante para la alta dirección, era poder monitorear el funcionamiento y los resultados financieros de las diferentes unidades de negocios para controlar las actividades diarias y hacer planes efectivos. Esto involucraba la preparación y distribución de los reportes de los resultados de las oficinas internacionales de Ericsson para cientos de administradores.

El intercambio de información dentro de la organización era informal. Se usaban diferentes medios, como el correo tradicional, el correo electrónico y el fax. No había una base de datos central que sirviera como referencia y las comunicaciones eran lentas y difíciles de manejar. El acceso de la alta dirección y la visión de negocios significativos y de indicadores financieros, eran inadecuados.

Los retrasos en los envíos de información y de respuestas a las consultas eran

inaceptables. Las prácticas de comunicaciones tradicionales usaban la tecnología de forma mecánica y sin imaginación, no se explotaban las ventajas de Internet ni las técnicas más recientes de bases de datos para mejorar el acceso de la alta dirección a la información crítica. Los problemas eran la falta de un formato común para enviar y recibir la información, para accederla, y la lentitud de la transmisión de datos.

Ericsson necesitaba un nuevo enfoque con el que los administradores financieros pudieran ver y examinar instantáneamente y desde cualquier lugar, el funcionamiento financiero y de negocios por país, tipos de negocios y de productos. La empresa quería ser capaz de medir su funcionamiento de acuerdo a sus indicadores estándares.

Había la necesidad de consolidar las estadísticas de forma centralizada, sintetizar los pronósticos, y poder planear desde cualquier lugar, usando la tecnología de Internet. El objetivo era que los administradores pudieran acceder a la información rápida y fácilmente para poder dar seguimiento del funcionamiento del negocio.

Ericsson escogió a Adaptive Server Enterprise y a EAServer de Sybase como las tecnologías base para su sistema de reportes y de manejo de sus bases de datos internacionales. Este fue desarrollado por Aaro Systems AB, su socio sueco de desarrollo de sistemas. Se les pidió que les proporcionaran a los administradores de Ericsson la información de los negocios actuales y del funcionamiento financiero, lo cual abarca los detalles de los ingresos de la hoja de balance, la información gráfica, y una tarjeta de seguimiento del funcionamiento. Todo ésto se ejecutaría en un ambiente seguro de Intranet, en el que los administradores podrían acceder a la información y responder a través de navegadores comunes como Netscape y Explorer.

"Anteriormente, el problema principal era ponerse de acuerdo en los formatos de reportes internos, comunes para los datos importantes", dijo Richard Smedberg, controlador financiero en Ericsson. "Esto provocó retraso en la recepción de la información relevante y en la asignación del acceso individual autorizado." También había problemas de comunicación con los administradores en algunas partes del grupo Ericsson. Como resultado, el flujo de información a través de la organización era lento.

El sistema piloto se terminó en tres semanas, el sistema completo se desarrolló en menos de tres meses y se instaló en todas las oficinas internacionales de Ericsson. "El usuario está conectado a la aplicación por medio de una interfaz de Internet, por ejemplo, Netscape o Explorer", dice Smedberg. "Se permite el acceso a cada usuario, dependiendo de su nivel de autorización y de sus necesidades individuales. Alrededor de 400 administradores en diferentes partes del mundo, acceden a la base de datos administrativa, pero muy pocos han usado el sistema completo."

- ***Comentarios***

En este caso se describió la experiencia que, de nueva cuenta, tuvo la compañía Ericsson, la cual pudo desarrollar con rapidez una aplicación que funciona en Internet y que utiliza una tecnología EAServer que contiene a Jaguar CTS.

V.4.2 Laboratorio Nacional Los Alamos

El Laboratorio Nacional Los Alamos (LANL) ubicado en Nuevo México, tiene un papel importante para la seguridad nacional de los Estados Unidos. Fue fundado durante la segunda guerra mundial para la investigación de armas nucleares. Actualmente se encarga de evitar la proliferación de armas nucleares y de organizar sus reservas.

LANL tiene dos tipos de personas involucradas en el proceso de adquisiciones: Los requisitores, quienes ordenan los bienes y servicios, y los especialistas en adquisiciones, que localizan los productos clave, negocían y administran los contratos, y se relacionan con los proveedores. LANL recientemente implementó el Escritorio de Adquisiciones, que es un sistema cliente/servidor para los especialistas en adquisiciones.

Para incrementar la eficiencia del proceso de negocios y del flujo de trabajo, LANL quiso mejorar su sistema de adquisiciones para que los usuarios pudieran enviar sus órdenes, revisar su estado y obtener las aprobaciones necesarias, adecuando el sistema de adquisiciones existente para trabajar en la Web.

El objetivo era mejorar el proceso de adquisiciones, permitiendo que los usuarios enviaran y dieran seguimiento a sus propias órdenes. Las soluciones fueron PowerBuilder, PowerJ y Jaguar CTS.

LANL seleccionó a Java como el lenguaje de desarrollo. "Había dos buenas razones para cambiar a Web el sistema de adquisiciones existentes usando Java", dijo el líder del equipo de infraestructura Web de LANL. "Primero, se tienen 7,000 usuarios que trabajan en muchas máquinas clientes, incluyendo Windows, Machintosh y Unix. Es un reto desarrollar un software cliente que se ejecute en las distintas plataformas. Así que, Java, con su independencia de plataforma, es una solución excelente. Segundo, se obtiene el beneficio de reducir el costo del desarrollo de aplicaciones al usar clientes delgados."

El *front-end* basado en Web opera a través de la Intranet de LANL usando una verdadera arquitectura de cliente delgado, por lo que ninguna de las funcionalidades es almacenada en la máquina cliente. La arquitectura Web consiste de un applet desarrollado con PowerJ, que se conecta a la capa intermedia de la aplicación que está escrita con PowerBuilder. Esta aplicación distribuida se conecta a la base de datos del sistema de adquisiciones en una IBM SP2 que ejecuta las bases de datos DB2 de IBM.

Debido a que Java es independiente de plataforma, la aplicación se puede ejecutar en cualquier cliente o servidor. Como resultado, los usuarios sólo necesitan un navegador que ejecute Java, como Netscape o Internet Explorer, para solicitar los servicios. Los usuarios simplemente introducen la información de la orden, como artículos en línea y los datos de consolidación, directamente a través de la interfaz Web. Y pueden hacer un seguimiento electrónico de la solicitud, para su aprobación y adquisición.

Se buscaba construir una aplicación empresarial con el potencial de Java y al mismo tiempo aprovechar la aplicación PowerBuilder existente. PowerJ fue la herramienta escogida porque ofrecía la arquitectura para la integración con PowerBuilder distribuido y herramientas robustas para las aplicaciones de la empresa. PowerJ es ideal para construir aplicaciones tipo Web con facilidad. "La habilidad y esfuerzo requeridos para desarrollar sistemas Java en PowerJ era más parecido al necesitado con una herramienta 4GL que al de una herramienta 3GL", dijo el director de LANL. "PowerJ es intuitivo y con un ambiente fácil de usar para el usuario, lo cual disminuye la curva de aprendizaje".

Los resultados fueron los siguientes:

- El Laboratorio Nacional Los Alamos pudo implementar su aplicación de adquisiciones en la Web.
- La nueva aplicación racionaliza el proceso de adquisiciones y reduce el costo al automatizar el proceso de negocios y de flujo de trabajo.
- Jaguar proporcionó la escalabilidad necesaria para que la aplicación funcionará en la Web.

LANL tiene ahora una aplicación de adquisiciones poderosa que funciona en la Web, racionaliza el proceso y capacita a los usuarios finales. "Esta solución permitirá que LANL reduzca los costos de administración de adquisiciones de bienes y servicios al automatizar el proceso de negocios y el flujo de trabajo". Dijo el especialista en contratos de LANL. "Lo más importante es que la solución permitirá a los especialistas en adquisiciones enfocarse en actividades de más valor y mejorar el servicio personalizado al unir fuertemente las adquisiciones con los requisitores". Además, recientemente se hicieron prototipos del uso de Jaguar en la capa intermedia de la arquitectura para probar la escalabilidad.

- ***Comentarios***

En este caso se obtuvo una aplicación que puede ser fácilmente escalable cuando las condiciones de uso así lo requieran, debido al uso de Jaguar.

V.4.3 Nordlandsdata : Comercio electrónico para el cine

La compañía de software noruega Nordlandsdata, utilizó EAServer para unir los sistemas de cómputo de cientos de cines sobre la Internet para liberar el primer sistema electrónico de boletaje de cine a nivel mundial. Desde su oficina matriz en Bodo, una ciudad con 40,000 personas en el norte de Noruega, Nordlandsdata ha captado la mayor parte del mercado de aplicaciones de boletaje de eventos.

El producto de la compañía, NDTS, es la aplicación de boletaje para cine usada por 75 cines en toda Noruega, la cual procesa ocho y medio millones de boletos de películas

cada año. La compañía empezó usando PowerStudio y Adaptive Server Enterprise de Sybase como ambiente de desarrollo de la nueva versión del NDTS para Windows NT. Su primer resultado fue la nueva versión para Windows de 32 bits. "Se probó Clarion, Microsoft, Delphi y Sybase, pero Sybase proporcionó la plataforma más completa para trabajar," dijo el Gerente de desarrollo de software.

El primer proveedor de software de Noruega para boletaje de cine, Nordlandsdata, fue propuesto por FilmWeb. Este es el sitio Web de películas noruegas perteneciente a los cines y teatros más importantes de ese país. La solicitud de FilmWeb era llevar su aventura Web al siguiente nivel, mediante el comercio en línea.

El reto para Nordlandsdata era construir una aplicación de mensajería en línea que uniera a los sistemas NDTS de los cines de Noruega con los sitios Web de comercio electrónico de FilmWeb, con el objetivo de ofrecer asientos en buenos lugares e información de la programación, así como la posibilidad de adquirir los boletos en línea. Para Nordlandsdata, este proyecto fue vital ya que la compañía obtuvo los derechos para vender la aplicación una vez terminada, y pudo desarrollar y probar un nuevo producto para Internet.

Esta vez, las soluciones fueron: EAServer, Jaguar CTS, Adaptive Server Enterprise, PowerBuilder, PowerPlus y PowerJ de Sybase.

Nordlandsdata necesitaba una plataforma para las aplicaciones Web, que proporcionara un punto de integración para muchos sistemas distribuidos, así como una plataforma que garantizara comunicaciones sólidas, transacciones robustas y alta disponibilidad.

"También se necesitaba un ambiente de desarrollo de software que diera soporte a las técnicas de desarrollo rápidas que ya se estaban usando con Powerstudio de Sybase. Esto era importante por razones de productividad en el desarrollo del software, para obtener una integración fácil con las aplicaciones existentes. Se necesitaba una aplicación Web que trabajara rápido," dijo el Gerente.

La compañía escogió a EAServer como el ambiente de desarrollo para su nueva aplicación. "Conocemos los productos de Sybase, los hemos usado y probado, y son muy buenos. También hemos recibido buenas referencias de los usuarios que ya están usando Jaguar CTS. Esto significa que no se tienen que buscar otros administradores de transacciones."

Ya que Jaguar proporciona un ambiente de ejecución para componentes de Java, COM y C++, los desarrolladores sólo tienen que hacer una aplicación en Java que contenga las reglas del negocio para la aplicación del servidor. "Jaguar redujo la complejidad de las tareas de programación", dijo el Gerente. "Su soporte multi-hilo significó que los programadores no tuvieron que escribir para un ambiente con hilos. Jaguar terminó con ésto."

Jaguar también se encargó de las comunicaciones entre la aplicación Web y los sistemas remotos NDTS, y proporcionó una liga al objeto CORBA desde las aplicaciones NDTS y la Internet. Al escoger EAServer, Nordlandsdata logró fácilmente la integración, haciendo una aplicación de procesamiento de transacciones altamente funcional en sólo dos meses. "Si no se tuviera Jaguar CTS, ésto habría tomado casi un año", dijo el Gerente.

Las ligas proporcionadas por Jaguar son responsables de mostrar la disponibilidad de asientos en los cines. Esta información es actualizada cada minuto y colocada en una base de datos Adaptive Server en la aplicación Web. Los usuarios pueden revisar la duración de las películas y ver al momento los asientos disponibles en su cine local. Cuando el cliente escoge una película, una hora y una sala, Jaguar le comunica esta información al cine a través de Internet, en donde es procesada por NDTS.

Jaguar se asegura de que sólo cuando la transacción de registro sea aceptada y guardada en la base de datos remota, un mensaje de confirmación sea enviado al usuario. El pago con tarjeta de crédito por la transacción es manejada por separado por EUNet, un proveedor alemán de servicio de Internet.

"No ha habido ningún problema con la parte de Nordlandsdata de la aplicación durante el contrato de tres meses que ofrecía el registro a los cines en Bodo", dijo el Gerente. FilmWeb se ha sorprendido por el funcionamiento y funcionalidaed de lo que es, en realidad, la primer versión de un sistema de procesamiento de transacciones distribuido de vanguardia. "Han sido muy positivos sobre la estabilidad y confiabilidad de EAServer", dijo el Gerente. Las pruebas de estrés realizadas al sistema han convencido al Gerente de que EAServer puede manejar cargas masivas eficientemente: "Este es un producto extremadamente estable. Podemos manejar muchas transacciones de boletaje en un minuto, sin fallas."

Gracias al uso de un protocolo de comunicaciones abierto, el sistema se ha extendido fácilmente para cumplir otro contrato, esta vez ofreciendo boletaje para el cine a subscritores con teléfono móvil. El promotor nacional de Noruega, Telenor Mobil, en conjunto con FilmWeb ofrece los servicios usando una cartera electrónica construida especialmente para teléfonos móviles.

Los clientes escogen una película, una sala y su asiento favorito, y pagan a través de su teléfono móvil. El cliente elige entre comunicarse mediante el servicio de mensajes cortos GSM hacia Telenor, o a través de Internet al sistema EAServer de Nordlandsdata. Una vez más, éste maneja las solicitudes de boletaje y une de forma efectiva al usuario con el cine.

Los resultados son que más de ocho milliones de boletos vendidos por los sistemas de Norlandsata para 75 cines. La elección de EAServer hizo cambios revolucionarios en los productos de Nordlandsdata, y le permitió atraer más ganancias, debido a los servicios que ofrece a los mercados de exportación.

Nordlandsdata planea sacar una versión gratuita para Internet de NDST para los cines y tomar sólo un pequeño porcentaje de cada transacción del boletaje, lo cual permitirá a sus clientes pagar, al mismo tiempo que ellos obtienen ganancias en línea. "Con 8.5 millones de boletos vendidos al año, pensamos que tal vez el 10% de éstos se venderán por Internet en los próximos diez años, así que ésta es una nueva fuente de ingresos". Dijo el Gerente.

Nordlandsdata está impulsando su propio mercado para el próximo año, pero la aplicación Web de la compañía ya está generando interés en Suiza y Dinamarca. "Queremos expandirnos, pero queremos hacerlo bien, ni tan rápido, ni tan lentamente. Con EAServer como nuestra plataforma Web, tenemos la certeza de que podemos cumplir las demandas futuras".

- ***Comentarios***

Otro de las ventajas de utilizar Jaguar se han observado en este caso, ya que la compañía Nordlandsdata ha obtenido mayores ingresos debido a que su sistema accede a la información y realiza transacciones en tiempo real, además de poder comunicarse con otros sistemas, sin importar la plataforma que éstos utilicen.

V.4.4 Deutsche Bank

Con los productos de Sybase, Deutche Bank creó un sistema de warrants[4] comerciales que funciona en tiempo real y a nivel global. Puede usarse localmente y extenderse a sus socios y clientes. Entre más turbulento es un evento en el mercado, es más riesgoso y se convierte en un campo fértil para el pago de warrants. El aumento de negociaciones de éstos, ha incrementado sus tarifas en los últimos años. Más de 7,000 warrants emitidos por las principales instituciones financieras se están negociando en los mercados, y se están comprando alrededor de 1,000 de las acciones listadas. El precio de esos derechos es mucho menor que el de las acciones, de los índices o de los futuros a los cuales están unidos. Por lo tanto, es posible que con una inversión relativamente pequeña, se muevan cantidades más grandes, lo cual es el motivo por el que los precios de los warrants reaccionan a las mas pequeñas fluctuaciones del mercado, incrementándose o cayéndose estrepitosamente.

Con 800 billones en activos, Deutsche Bank es el líder en el ambiente bancario. Pero también aspira a mejorar su posición en el nuevo y lucrativo mercado de rápido crecimiento de warrants, y ven a su sistema de negociaciones db STAR como una buena solución de Sybase para lograrlo.

Las soluciones utilizadas fueron EAServer, PowerJ, Jconnect y DirectConnect de Sybase. Los resultados fueron un sistema de comercio completamente automatizado

[4] Los warrants son contratos por los derechos de compra o venta de algún producto financiero derivado en cierto momento con un precio acordado.

que permite a los intermediarios y a los clientes institucionales negociar los warrants ofrecidos por Deutsche Bank a través de Internet, Intranet, o una línea de arrendamiento.

"Era importante para nosotros desarrollar un sistema altamente automatizado que garantizara cierta tarifa a nuestros clientes con precios en tiempo real, de forma que los warrants emitidos por nosotros fueran atractivos y fáciles de negociar", explica el administrador de proyectos de Deutsche Bank.

Las experiencias con la competencia mostraron que las buenas soluciones en las tecnologías de información usadas por los emisores de warrants tienen un efecto directo en el comportamiento de los intermediarios y de los inversionistas, y por lo tanto, en las acciones del mercado.

El banco decidió construir una aplicación de tres capas basada en la tecnología de Sybase y ACS Systemberatung GmbH como socios de desarrollo. Con más de una docena de conexiones a sistemas remotos, la aplicación propuesta iba a soportar comunicaciones masivas, por lo cual, la programación robusta y sencilla de EAServer fue muy atractivo para Deutsche Bank.

El componente central de dbSTAR es un sistema de precios en tiempo real, que es activado automáticamente por los reportes correspondientes de Reuters. Cada vez que se introduce un nuevo precio en las existencias, el sistema calcula los nuevos precios de los warrants correspondientes, en fracción de segundos. Lo cual hace que sean de gran importancia las funciones de mensajería del sistema.

"El cambio de los precios debe mostrarse directamente en la pantalla y los mensajes tienen que aparecer en las pantallas de los negociantes involucrados. Esto no sería posible si se usaran dos capas y no queríamos ir a expensas de programar una capa intermedia en el servidor de aplicaciones por nosotros mismos", dijo el administrador de proyectos de Deutsche Bank.

"Además de habernos convencido por la tecnología, una de las razones más importantes para tomar esta decisión, fue el hecho de que Sybase podía cubrir toda la funcionalidad necesitada: desde la capa de la base de datos y la capa intermedia, hasta las herramientas de front-end orientadas a Internet", dijo Lüders. Y el trabajo colectivo ha confirmado que la decisión tomada fue la correcta: "Todos los miembros del equipo han trabajado muy bien juntos."

Desde marzo de 1999, los negociantes locales han usando dbSTAR (Aplicación Segura de Negociación de Productos para Presupuestos Remotos) y ha logrado mayor productividad. Al mismo tiempo, BANK 24 se ha convertido en el primer banco directo para negociar mediante el nuevo sistema.

Se definieron más de una docena de interfases con sistemas internos y se implementaron mecanismos estrictos de seguridad. Esta parte del sistema fue

diseñado por completo por Sybase como parte de una arquitectura completa que se integra con los demás componentes, como las bases de datos, los sistemas externos como Reuters, y los clientes y las aplicaciones *front-end* desarrolladas por ACS con PowerJ.

Las bases de datos importantes tienen un sistema de documentación para el emisor de los nuevos warrants, EWMS (Sistema de Administración de Ordenes Electrónicas), y se cuenta con una base de datos maestra y sistemas de administración de riesgos, los cuales proporcionan los datos requeridos por el mercado para que dbSTAR calcule los precios.

En la primer etapa de expansión, el principal elemento técnico de la solución es un servidor de aplicaciones Jaguar CTS. Un segundo servidor será instalado en la oficina de Frankfurt de forma que se garantice la máxima disponibilidad del sistema en cualquier momento. En la segunda etapa, la misma configuración será instalada en Londres con los servidores de replicación de Sybase, para asegurar la consistencia de los datos.

Desde el momento en que el precio es dado, el cliente tiene pocos segundos para aceptar o rechazar el trato o la oferta. Si es aplicable, el sistema automáticamente genera la confirmación de la transacción y protege la transacción con sus sistemas de respaldo. Esto significa que el cliente recibe inmediatamente su reporte y puede ver las ganancias o los costos de la negociación.

Lüders también ve una ventaja atractiva en la batalla por las acciones en el mercado: "El cliente ya no tiene que esperar recibir la factura por correo, lo cual toma por lo menos un día. Ahora puede completar la transacción ese mismo día. Si es necesario, puede volver a comprar o vender tres veces en veinte minutos, mientras cada trato se hace en pocos segundos."

El registro de los datos necesitado por el sistema de administración no es introducido por ninguna persona y, dependiendo del tipo de warrants, se envían a las computadoras correspondientes en Frankfurt o en Londres. Además, asegura que las facturas para los clientes alemanes se establezcan dentro del sistema mainframe, de forma que se cumplan los mínimos requerimientos legales. El sistema de negociación también calcula automáticamente las operaciones alternas necesarias para asegurar el trato.

Automáticamente, el sistema calcula los precios y el contrato es confirmado con un click del ratón. El acuerdo interno entre los proveedores de servicios financieros involucrados también está automatizado. Los intermediarios tienen la opción de permitir a sus clientes el acceso directo a través de Internet. "Tener este sistema es una gran ventaja para Deutsche Bank, ya que representa un atractivo incentivo para que los intermediarios y los clientes paguen sus warrants".

Los clientes locales, que son los comerciantes del Deutsche Bank, pueden acceder a

toda la información que necesiten desde su pantalla. Cada usuario puede configurar su propio perfil de acuerdo a sus necesidades. La flexibilidad requerida para que los clientes externos se conecten, está garantizada porque Jaguar CTS soporta EJBeans y componentes compatibles con CORBA. Los clientes finales sólo pueden acceder a dbSTAR a través de los servidores de los intermediarios, ya que la responsabilidad de los intermediarios hacia los emisores es asegurarse de que sus clientes cumplan con los requerimientos legales para hacer transacciones de negocios. El diseño abierto del sistema significa que los intermediarios tienen la responsabilidad de la administración del otro lado del *firewall*.

Con las aplicaciones de Internet, los intermediarios y los corredores proporcionan una interfaz de usuario para negociar directamente desde su computadora personal los warrants publicados por Deutsche Bank.

Si un cliente llama al centro de llamadas de BANK 24 para comprar o vender warrants, los miembros del equipo pueden dar inmediatamente el precio actual, entrando al sistema con un código de identificación de seguridad o dando un click sobre el warrant. Y si el cliente coloca un warrant en estos términos, el negocio se hace simplemente dando un click con el ratón.

Pero dbSTAR puede hacer más, debido a que Deutsche Bank AG no sólo está enfocado a los bancos directos, a los corredores e inversionistas institucionales, sino también a atender al cliente final. Los bancos directos y los corredores pueden ofrecer acceso directo a los sistemas de negocios a través de sus servidores, como un servicio especial, gracias a la aplicación front-end basada en Java y a la automatización total.

Al decidir desarrollar el sistema con tecnología Java y con EAServer, Deutsche Bank asegura que su nuevo sistema esté preparado para el futuro. Aunque Java no garantiza ser por completo una solución "independiente del sistema operativo", los sistemas basados en él, van en esa dirección. Esto significa que los costos de instalación para el cliente permanecen bajos, lo cual es una gran ventaja para un proyecto global.

"Tenemos razones para creer que en dbSTAR hemos creado negocios de alta tecnología y sistemas de información, con lo que esperamos mantener y expandir nuestra posición en el mercado" concluye Lüders.

- ***Comentarios***

Una vez más, se observa la creación de una aplicación que funciona en tiempo real sobre Internet y que realiza interfases con otros sistemas, sin importar la plataforma que utilicen. El que las transacciones se realicen en tiempo real representa una gran ventaja en el ambiente financiero, ya que se tiene una mayor demanda, en este caso de warrants, lo que significa una mayor ganancia para todos.

V.4.5 Universidad de Berkeley

Los objetivos eran los siguientes:

- Proporcionar acceso a las colecciones de museos y a archivos de la Universidad de Berkeley mediante la Web.
- La creación de aplicaciones flexibles y reutilizables administradas centralizadamente en un ambiente rápido de desarrollo.
- La migración eficiente de las aplicaciones existentes de dos capas que usaban XWindows a aplicaciones Web de Java de tres capas.
- Como solución, se usaron AEServer 11.5, Jaguar CTS, PowerJ y jConnect de Sybase.

Los resultados fueron:

- Todo el mundo puede acceder a las bases de datos de la colección de más de un millón de registros ligados a miles de imágenes digitales.
- Desarrollo rápido de aplicaciones Web para un número creciente de colecciones, con el uso de componentes "escritos una vez, ejecutados en donde sea", que son administrados y activados en tiempo de ejecución.
- Nominación en 1998 para el premio Computerworld Smithsonian por ser una aplicación innovadora de la tecnología de la información.
- En 1868, la Universidad de California fue nombrada para administrar la colección estatal de los especímenes geológicos y biológicos, y de organizar y presentar las colecciones del museo.

"La colección ha crecido y las herramientas y técnicas para administrarlas han evolucionado rápidamente", dijo el director del Proyecto de Información del Museo (PIM) de la Universidad.

El sitio Web del PIM incluye museos, archivos departamentales y otras fuentes. Con sólo un click se pueden obtener imágenes de la colección, texto y otras ligas relacionadas.

"El enfoque de componentes ha sido excelente para nosotros debido a que se tienen numerosos y diferentes proyectos en el campus", explica el director.

"Primero se creó un conjunto de componentes y se unieron en distintas combinaciones, dependiendo del proyecto. Esto es más eficiente que cada vez crear una aplicación nueva. En segundo lugar, se puede administrar centralizadamente el acceso al Web, lo cual es más eficiente. Para las colecciones se pueden diseñar bases

de datos que cumplan con sus necesidades, y los usuarios finales pueden estar en la plataforma que quieran, mientras se mantiene y actualiza la capa de acceso".

Otras ventajas del enfoque de componentes son, que a corto plazo el PIM puede desarrollar una nueva funcionalidad más rápidamente. A largo plazo, facilitará el soporte de las bases de datos institucionales.

- ***Comentarios***

En este caso se observa que el enfoque orientado a componentes ayudó a la migración de la aplicación de forma rápida, y a contar con una administración centralizada de dichos componentes, los cuales pudieron ser compartidos con otros proyectos para evitar repetir la codificación de ciertos procesos.

Conclusiones

Actualmente, las tecnologías de información deben permitir realizar cambios significativos a los sistemas corporativos, de forma rápida y efectiva.

En México, la mayoría de los negocios se encuentran trabajando con el enfoque de dos capas y algunos ya están migrando a sistemas de tres capas. En el presente trabajo se han explicado las razones por las que la arquitectura de dos capas presenta limitaciones, aunque dependiendo del tipo de aplicación a desarrollar, será el enfoque utilizado.

Un sistema de tres capas, inicialmente cuesta más que uno de dos, debido a que requiere de mayor trabajo de planeación y programación, pero finalmente resulta ser más escalable y efectivo en cuanto a costos a largo plazo.

Por otra parte, operar desde Internet se está convirtiendo en un nuevo e importante tipo de cómputo distribuido. La Internet extiende los beneficios de la computación cliente/servidor y gracias a los estándares establecidos, se ha ampliado el uso de este enfoque.

El número y la variedad de aplicaciones en Internet está aumentando, y Jaguar CTS, así como otras tecnologías de red, están facilitando el despliegue de aplicaciones que manejan transacciones en la red.

Actualmente, se realizan negocios por Internet, por lo cual son necesarias las aplicaciones con contenido dinámico, acceso seguro a bases de datos y procesamiento de transacciones a gran escala.

Como se ha visto, Jaguar es un servidor de transacciones de componentes diseñado para construir aplicaciones multi-capas de negocios, que cuenta con la tecnología necesaria para desarrollar con facilidad y eficiencia este tipo de aplicaciones para la Web.

Además, combina las mejores características de un ORB y de un monitor de procesamiento de transacciones basado en componentes, cuenta con un núcleo multi-hilos que hace más rápidas a las aplicaciones, trabaja con todos los modelos líderes de componentes, simplifica la instalación y la configuración de las aplicaciones basadas en transacciones, y está orientado al cómputo abierto.

Como se mostró en el presente trabajo, actualmente varias empresas internacionales, ya han tenido la necesidad de utilizar una herramienta que les permita *manejar transacciones a gran escala a través de la Web*, y su experiencia con Jaguar CTS ha sido satisfactoria, por lo que éste ha probado ser una buena opción para el desarrollo de este tipo de aplicaciones.

El uso de Jaguar puede ser una gran ventaja para las empresas cuyo estándar de manejadores de bases de datos sea Sybase, debido a que ya cuentan con el contacto y con la experiencia de haber trabajado en esa plataforma. También lo es, para las empresas que quieran utilizar Sybase como manejador de bases de datos, pero que todavía cuentan con aplicaciones hechas en diferentes lenguajes de programación y que no quieran perder dinero y tiempo en migrar sus aplicaciones actuales a un mismo lenguaje.

Jaguar CTS permite reducir el tiempo de desarrollo y el costo total del diseño de las aplicaciones, y al mismo tiempo, soporta los estándares de protocolos de seguridad, como HTTPS.

Si una compañía quisiera empezar a desarrollar con lenguajes Java o C, tendría que contar con programadores que tuvieran conocimiento o experiencia suficiente sobre dichos lenguajes, de otra forma, ésto resultaría ser más costoso en tiempo y dinero.

Como se ha mostrado, Jaguar tiene las siguientes ventajas, que se deben tomar en cuenta al desarrollar un sistema:

- Ambiente gráfico.
- Uso de componentes hechos con distintos lenguajes.
- Sigue el estándar CORBA ORB.
- Maneja seguridad.
- Maneja hilos.
- Coordina transacciones.

Durante el desarrollo de la aplicación, los programadores pueden compartir mediante la red los componentes existentes y de esta manera evitan repetir código con la misma funcionalidad.

Las desventajas que puede tener Jaguar, son:

- Al ser recomendable programar con Java para crear componentes, el uso de éste es más complejo que el de otros lenguajes, además de que la identificación correcta de todos los archivos de clases que se utilizan puede resultar un trabajo engorroso.

- En México no existe un buen soporte por parte de Sybase sobre Jaguar.

- Si ocurre algún error, éste de debe buscar en tres capas y no sólo en dos.

En la comparación realizada entre varios servidores de transacciones, se concluyó que Jaguar era el único que cumplía con el estándar CORBA y que aceptaba el mayor tipo de objetos.

El presente trabajo ha pretendido ser una guía para entender el funcionamiento del servidor Jaguar CTS, así como de la creación de sus componentes, ya que actualmente en el mercado no existe mucha bibliografía, especialmente en español, sobre el tema.

Finalmente, esta tesina es mi manera de proporcionar más información a los estudiantes sobre uno de los tópicos actuales de las tecnologías de arquitecturas distribuidas, ya que si México también quiere entrar en la competencia de aplicaciones en la red, la industria nacional debe tener a su alcance el conocimiento de dichas tecnologías de punta, que le permita obtener el nivel necesario para ocupar un buen lugar en el mercado internacional, y es a través de los próximos profesionistas que logrará estos objetivos.

Glosario

ActiveX
Es la respuesta de Microsoft a Java. Es una implementación de OLE diseñada para ejecutarse en ligas de Internet.

Adaptive Server Enterprise
Es el servidor en la arquitectura cliente/servidor de Sybase. Maneja muchas bases de datos y usuarios. Es responsable de localizar los datos en los discos, dirige la descripción de datos lógicos a los almacenamientos físicos, y mantiene los datos y los procedimientos en cachés de memoria.

API
(Application Program Interface) Interfaz de Programa de Aplicación. Es una interfaz entre el sistema operativo y los programas de aplicación. Incluye la forma en que los programas de aplicación se comunican con el sistema operativo, y los servicios que el sistema operativo tiene disponibles para los programas. Por ejemplo, una API puede hacer posible que programas que se ejecuten bajo ésta, abran Windows y muestren ventanas de mensajes.

Applet
Es un programa de aplicación escrito en Java que puede ser descargado de un servidor Web y ejecutado por un navegador. El navegador descarga el applet al igual que lo hace con un archivo gráfico. Por seguridad, el applet no puede acceder los archivos del sistema del cliente en el que se está ejecutando y la comunicación del applet a través de la red es limitada al servidor desde el cual se descargó.

ARP
(Adress Resolution Protocol). Protocolo de Resolución de Direcciones. Es un método para encontrar la dirección Ethernet de un host a partir de su dirección de Internet. El emisor transmite un paquete ARP que contiene la dirección de Intenet de otro host y lo espera para devolver su dirección Ethernet.

Arquitectura Distribuída
Conjunto de componentes diseminados en una red que, como un todo, responden a un requerimiento.

Asíncrono
Se llama así a un proceso en un sistema multi-tarea cuya ejecución puede hacerse independientemente, en background. Otros procesos pueden empezar antes de que el proceso asíncrono termine.

ASP
(Active Server Pages) Páginas de Servidor Activas. Es un ambiente para scripts usados en Information Server de Microsoft, en el cual se pueden combinar páginas HTML, scripts y componentes reutilizables de servidores ActiveX para crear páginas Web dinámicas.

ATM
(Asynchronous Transfer Mode). Modo de Transferencia Asíncrono. Es un método para asignar dinámicamente el ancho de banda usado por un paquete de tamaño fijo.

AWT
(Abstract Window Toolkit). Conjunto de herramientas de Ventanas Abstractas. Son herramientas gráficas de Java para la interfaz de usuario y un sistema de ventanas independiente de plataforma. AWT es parte de la JFC (Java Foundation Classes), que es la API estándar que proporciona una GUI para un programa escrito en Java.

AXE
Editor de texto para el sistema X Windows.

Back-end
Es cualquier software que realiza, ya sea la etapa final en un proceso, o una tarea que el usuario no ve. Un uso común es un compilador. El back-end del compilador genera un lenguaje máquina y realiza optimizaciones específicas de la arquitectura de la máquina. El término también se puede usar en el contexto de aplicaciones de red. Ejem. "El back-end del sistema maneja protocolos de sockets".

Back-ground
Una tarea que corre en back-ground es separada de la terminal en donde empezó, y frecuentemente se ejecuta con una prioridad más baja.

Base de Datos Orientadas a Objetos
Aplicación que permite el almacenamiento y el manejo de la información mediante el modelo de objetos.

BPR
(Business Process Reengineering). Reingeniería de Proceso de Negocios. Son técnicas de análisis de procesos de negocios y de flujos de información para optimizar los costos de producción.

Built-in
También conocido como primitiva. Una función u operador built-in es proporcionado por el nivel más bajo de la implementación de un lenguaje. Usualmente no es posible expresarlo con el mismo lenguaje.

Bus
Es un enlace o conductor común; es un método de interconexión de dispositivos mediante una sola línea compartida.

Caché
Estructura que almacena resultados de las últimas solicitudes para evitar el costo de volver a calcularlos. En un navegador, el caché guarda copias de los documentos que acceden más frecuente en el disco duro, para que al volverse a cargar, aparezcan más rápidamente.

Capa
Las capas son niveles dentro de la estructura de una red, que proporcionan ciertos servicios a otras capas para ocultando los detalles de éstas.

Capa de Negocios
En una arquitectura multi-capa, es la que contiene a los componentes que contienen la lógica del negocio.

Cast
Sintaxis de programación para especificar que el valor de una expresión se debería convertir a un tipo de dato diferente.

CICS
(Customer Information Control System). Sistema de Control de Información Personalizada. Es un sistema de comunicación de IBM que maneja bases de datos.

Clase
En la programación orientada a objectos, es un modelo o plantilla que puede ser instanciada para crear objectos con una definición, propiedades, operaciones y comportamiento comunes.

CLI
- (Command Line Interfase). Interfaz de Línea de Comando. Es un medio de comunicación entre un programa y el usuario, basado en una entrada y una salida textual. Los comandos son introducidos por el teclado o con dispositivods similares y son interpretados y ejecutados por el programa. Los resultados salen como texto o gráficos en la terminal. La CLI usualmente provee mayor flexibilidad que la GUI, con la desventaja de ser más difícil de usar.
- (Call Level Interfase). Interfaz a nivel llamada. Es una interfaz de programación diseñada para soportar acceso SQL a bases de datos.

Cliente Delgado
La mayor parte del procesamiento se ejecuta en el servidor y no se requiere de muchos recursos en la computadora del cliente.

Cliente/Servidor
Una forma común de sistemas distribuidos en el cual el software es dividido entre las tareas del servidor y las del cliente. Un cliente envía una solicitud a un servidor mediante algún protocolo, y el servidor le contesta.

Código
Son instrucciones de programación de computadoras.

COM
(Component Object Model) Modelo de Objetos de Componentes. Es un modelo para sistemas distribuidos, creado por Microsoft.

Commit
Instrucción de Sybase SQL que ejecuta por completo las transacciones abiertas.

Componente
En Jaguar, es un "objecto de aplicación" que consiste de uno o más métodos. Comúnmente ejecutan la lógica del negocio, acceden las fuentes de datos, y devuelven resultados al cliente. Los clientes crean una instancia de un componente y ejecutan sus métodos.

Concentrador
Es un dispositivo que combina los flujos de datos de muchas entradas simultáneas en un canal compartido de forma que se puedan separar después de la transmisión. El ancho de banda de la salida del concentrador debe ser al menos del mismo tamaño que el de las entradas simultáneas. Es una especie de dispositivo multiplexor.

Conjunto de Resultados
Renglones de datos regresados de la base de datos cuando se ejecuta una sentencia de SQL SELECT. El valor de regreso de la llamada a un método.

CORBA
(Common Object Request Broker Architecture). Tecnología de objetos distribuidos multi-lenguaje y multi-plataforma.

CPU
(Central Process Unit). Unidad Central de Proceso. Es la parte de la computadora que controla a todas las demás. En general consiste de una unidad de control, una de lógica y aritmética y otra de memoria.

Cross-platform
(Plataforma Cruzada). Indica que un programa está disponible para más de un tipo de computadora. Por ejemplo, un programa cross-platform debe estar disponible para PC, OS/2, y Macintosh.

CTS
(Component Transaction Server). Servidor de Transacciones de Componentes. Es el software que proporciona el servicio de desarrollo y administración de la capa intermedia para aplicaciones distribuidas basadas en componentes.

Datagrama
Es una entidad de datos contenida en sí misma e independiente, que contiene la información suficiente para ser transmitida de la computadora fuente a la destino sin seguridad en los intercambios entre éstas y la red de transporte.

Datawindows
Es un objeto visual de PowerBuilder de Sybase, que define la fuente de los datos y el estilo de presentación de éstos.

DBMS
(Data Base Management System) Sistema Manejador de Bases de Datos. Es un conjunto complejo de programas que controlan la organización, almacenamiento y recuperación de datos para muchos usuarios. Ejemplos de DBMSs son Oracle, Sybase y Datacom.

dbQ
Encolamiento de bases de datos. Son colas residentes en la base de datos que aseguran la integridad de las transacciones

Despliegue
Acción de mandar componentes a un contenedor, como Jaguar o PowerDynamo.

DLL
(Dynamic Link Library). Librerías dinámicas utilizadas para aplicaciones de MS Windows.

Dominio
Es una dirección de Internet en forma alfabética. Los nombres de dominios deben tener al menos dos partes: la izquierd, que indica la organización; y la derecha, que identifica el subdominio más alto, como el país (fr parar Francia, uk para Reino Unido) o el tipo de organización (com para comercial; edu para educacional, etc.).

Downsizing
Reducción de una estructura.

DSS
(Decision Support System). Sistema asistente en la toma de decisiones.

DTC
(Distributed Transaction Coordinator). Coordinador de Transacciones Distribuidas.

EAServer
(Enterprise Application Server) Servidor de Aplicaciones Empresariales. Es un conjunto integral de servidores de aplicaciones que se usan para desarrollar aplicaciones Web que soportan un volumen grande de tráfico, contenido dinámico y procesamiento intenso de transacciones en línea. Consiste de Jaguar CTS, Power Dynamo (Servidor de páginas dinámicas), Integrador de aplicaciones, Adaptive Server Anyware (manejador local de bases de datos) y PowerJ.

EDI
(Elecronic Data Interchange). Intercambio Electrónico de Datos. Aplicación para el intercambio de documentos y datos entre empresas, que permite hacer transacciones por Internet. Complemento del Comercio Electrónico entre Empresas o B2B.

Encapsulación
1. Técnica usada por los protocolos de capas en la cual una capa agrega la información del encabezado a la unidad de datos de protocolo (PDU) de la capa de abajo.
2. Es la forma de esconder los procesos internos de un conjunto de funciones de una interfaz. Es la técnica para mantener juntos a las estructuras de datos y a sus métodos.

Escalabilidad
Es la capacidad de un sistema para adaptarse a tiempo a una mayor o menor intensidad de uso, volumen, o demanda.

Esqueleto
Es una clase en el servidor que permite a un ORB hacer solicitudes a un objeto, del cual no se conoce su implementación al tiempo de compilación. Actúa como interfaz entre Jaguar y el código que implementa el método. Los esqueletos son compilados y ligados a cada componente, y en tiempo de ejecución le permiten a Jaguar localizar e invocar al método apropiado. Los esqueletos siempre está en el lado del servidor.

Esquema
Conjunto de sentencias expresadas en un lenguaje de definición de datos, que describen completamente la estructura de una base de datos.

Ethernet
Es una red de área local.

Evento
Es algo que pasa y que tiene algún significado para una tarea o programa, como la terminación de una operación de entrada/salida.

Excepción
Condición anormal en el ambiente Java, como un error desconocido de comunicación.

Fan-in/Fan-out
Son equipos que permiten dividir la señal del mismo a varios equipos.

FIFO
(First-in-First-out). Es una estructura en la cual los datos son sacados en el mismo orden en que se metieron.

Firewall
Mecanismo de seguridad que impide el acceso a una red.

Firma
Es la descripción de los argumentos de un función o método.

Frame Relay
Es la especificación de una interfaz que surgió como resultado para los requerimientos de velocidad en una red de área ancha; para la intercomunicación de LAN's y WAN's y de LANS' y LAN's; las comunicaciones con gran cantidad de datos; la multiplicidad de protocolos y la transparecia de éstos.

Framework
Es un conjunto de clases que cooperan entre sí, formando un diseño reusable. Define la arquitectura de la aplicación y la colaboración entre clases, lo cual le permite al diseñador de la aplicación concentrarse en los detalles específicos de la misma.

Front-end
Se refiere a las aplicaciones de usuarios finales. Puede tratarse de una computadora intermedia que hace actualizaciones y filtrados para otra (usualmente más poderosa pero menos amigable) máquina ("back end").

Funneling
Tunelaje. Proceso en el que la información es enviada a su destino en cuanto está disponible.

Gateway
Puerta de acceso o pasarela. Es el punto de enlace entre dos sistemas de redes.

GUI
(Graphical User Interface). Interfaz de Usuario Gráfica. Contiene gráficos, ventanas, íconos, menus, etc.

Hacker
Persona que se divierte explorando los detalles de los sistemas programables.

HFS
(Hierarchical File System). Sistema Jerárquico de archivos. Es un sistema en el que los datos son almacenados jerárquicamente en directorios y subdirectorios (como en DOS), o carpetas dentro de carpetas (como en Macintosh). La mayoría de los sistemas operativos tienen este sistema.

Hilo
Es la unidad más básica de un sistema operativo, capaz de ejecutar instrucciones en el CPU. Representa un camino de ejecución y tiene su propio espacio de direcciones y su propia pila de programación. Todo proceso tiene al menos un hilo, el hilo primario, el cual se inicia automáticamente con la aplicación.

Host
Es la máquina en la que reside una base de datos, una aplicación, o un programa. En TCP/IP, es cualquier sistema que está asociado con al menos una dirección de Internet.

HTML
(Hypertext Markup Language). Lenguaje usado para escribir documentos para servidores Web.

HTTP
(Hyper Text Transfer Protocol). Protocolo de transferencia de archivos usado en Internet.

HTTPS
Es una variante de HTTP para manejar transacciones seguras, conecta a servidores HTTP usando SSL.

IDE
(Interactive Development Environment). Ambiente de desarrollo interactivo. Es un sistema para escribir software, que tiene un editor de sintáxis, herramientas gráficas y soporte integrado de compilación y ejecución.

IDL
(Interface Definition Language). Lenguaje de Definición de Interfaz. Es un lenguaje definido por la OMG para crear la especificación de un componente, así como su interfaz; y de esta forma pueda interactuar con otros componentes. No provee detalle de la implementación, es puramente declarativo.

IIOP
(Internet Inter-ORB Protocol). Es un protocolo de comunicación CORBA. Está orientado a objetos y hace posible que los programas distribuidos escritos en diferentes lenguajes de programación se comuniquen sobre la Internet.

IMS
(Information Management System). Sistema de Administración de Información. Es un sistema de bases de datos de IBM.

Instancia
Es un elemento creado a partir de una clase.

Instanciar
Es crear un elemento (instancia) a partir de una clase.

Intranet
Tecnología de desarrollo de aplicaciones que utiliza los conceptos de Internet en una red privada.

ISAPI
(Internet Server Application Programming Interface). Interfaz de Programación para Aplicaciones de Servidores de Internet.

IT
(Information Technology). Tecnología de Información. Tecnología de datos que maneja el procesamiento de la información.

Jaguar CTS
(Jaguar Component Transaction Server). Servidor de Transacciones de Componentes. Proporciona un marco para el despliegue de la lógica de la capa intermedia o de las aplicaciones distribuidas basadas en componentes.

Jaguar Manager
Es el entorno gráfico del EAS, que se utiliza tanto para configurar el servidor Jaguar, como para definir y administrar componentes y paquetes dentro del mismo.

JAR
Es un archivo compuesto por otros archivos.

Java
Lenguaje de programación orientado a objetos que genera código portable que soporta la interacción entre objetos remotos. Java fue desarrollado por Sun Microsystems.

JavaBeans
Es la arquitectura de componenrtes independientes de plataforma para Java. Permite unir las partes de código Java en un ambiente de desarrollo drag-and-drop.

JavaScript
Es un lenguaje para scripts que se parece a Java y fue desarrollado por Netscape.

jConnect
Es el manejador de Sybase para JDBC que está hecho con Java y puede ser usado por applets. También lo pueden usar Sun y las máquinas virtuales de Microsoft.

JDBC
(Java Database Connectivity). Es una interfaz de programación de la aplicación que tiene las mismas características que un ODBC, pero es usada específicamente por las aplicaciones Java a bases de datos.

JVM
(Java Virtual Machine). Es un ambiente de ejecución para programas Java. Las JVMs pueden ser stand-alone (separadas) o estar integradas en los navegadores Web, servidores de transacciones, sistemas manejadores de bases de datos, etc.

LAN
(Local Area Network). Red de Area Local.

Legacy Systems
Sistemas corporativos establecidos de acuerdo a las reglas de operación de una empresa.

Librería de tipo
Librería que contiene las declaraciones de tipos de datos necesarios para una aplicación.

Mainframe
Originalmente significaba el gabinete que contenía al CPU de una computadora muy grande. Después de que las minicomputadoras estuvieron disponibles, se refería a la misma computadora grande. Actualmente es la estructura principal, una computadora grande tipo multi-usuario.

Marshal
Es la acción de copiar datos de manera apropiada para ser usados por otro objecto. Los stubs realizan el marshalling.

Método privado
Es aquel que sólo puede ser accedido por el objeto en donde se encuentra el método.

Método protegido
Es aquel que sólo puede ser accedido por el objeto en donde se encuentra y por los descendientes de éste.

Método público
Es aquel que puede ser accedido por cualquier otro objeto.

Mobile
Tecnología con software portatil.

MRE
(Managed Report Environment). Entorno de generación, distribución y planificación de reportes.

Multi-Capa
(Multi-Tier). Esquema con varias capas de software con responsabilidades bien delimitadas.

Multi-hilo
(Multi- threading). Posibilidad de crear varias secuencias de ejecución simultáneas dentro de un mismo proceso.

Multi-Plataforma
Característica de las aplicaciones de ejecutarse en ambientes distintos sin necesidad de reescribir código.

Navegador
(Browser) Programa cliente que solicita consultas a un servidor Web y despliega la información que éste le regresa.

Normalización
Se dice que una tabla en una base de datos relacional está normalizada si satisface ciertas restricciones.

NSAPI
(Netscape Application Programming Interface). Interfaz de Programación de Aplicaciones para Netscape.

Objecto
Es una unidad de datos con instrucciones para las operaciones que se realizan en ésta.

ODBC
(Open Data Base Connectivity). Conexión a base de datos abierta. Es una API estándar para acceder a datos en DBMSs relacionales y no relacionales. Usando esta API, las aplicaciones de bases de datos pueden acceder a datos almacenados en DBMSs en varias computadoras, aunque cada DBMS use un formato de almacenamiento de datos

y una interfaz de programación diferentes.

OLE
(Object Linking and Embedding). Es un sistema de objetos distribuidos y protocolos de Microsoft.

OLTP
(On Line Transaction Processing). Procesamiento de transacciones en línea.

OMG
(Object Management Group). Grupo de Administración de Objetos. Grupo formado por más de 700 empresas del ambiente informático.

OOP
(Object Oriented Programming). Programación Orientada a Objetos.

ORB
(Object Request Broker). Componente de software que permite que objetos CORBA operen entre sí, aunque estén en distintas computadoras.

OSI
(Open Systems Interconnect). Interconexión de Sistemas Abiertos.
Modelo de arquitectura de red y conjunto de protocolos que la implementan, desarrollada por ISO en 1978 como un marco de trabajo para estándares internacionales para una arquitectura de red con computadoras heterogéneas. Está dividida en siete capas: física, de unión de datos, de red, de transporte, de sesión, de presentación y de aplicación.

Patrón de Diseño
Es la descripción de un problema que ocurre en forma recurrente, y de su solución, de forma que ésta pueda aplicarse a situaciones similares.

Paquete
En Jaguar, es una colección de componentes que trabajan juntos para proveer un servicio o la lógica de negocios. Todo paquete actúa como una unidad de distribución, agrupando de manera conjunta los recursos de una aplicación para facilitar su despliegue y administración.

PDF
(Portable Document Format). Formato de Documento Portable.

Pila
Es una estructura para almacenar datos que se acceden en orden del último en entrar, primero en salir.

Plug – in
Hardware o software que inmediatamente después de haber sido instalado, puede usarse, en oposición al hardware o software que requiere de configuración.

Poleo
(Pooling). Es la forma de designar a un conjunto de recursos de uso común que pueden ser accedidos y reutilizados.

PowerBuilder
Lenguaje de cuarta generación basado en objetos, de Sybase.

PowerDynamo
Servidor de páginas dinámicas, de Sybase.

Primitiva
Es una función, operador o tipo de dato, dentro de un lenguaje de programación (o sistema operativo), para hacer más rápida la ejecución o porque es imposible escribirla en el lenguaje. Las primitivas comúnmente incluyen las operaciones lógicas y aritméticas (más, menos, y, o, etc.) y se implementan con pocas instrucciones de lenguaje máquina.

Proceso
Es un programa de ejecución. Consiste del código del programa (que puede ser compartido con otros procesos que se están ejecutando en el mismo programa), y algunos datos privados.

Proxy
Programa que actúa como intermediario entre los usuarios e Internet. Garantiza la seguridad, el control administrativo y el almacenamiento temporal de las páginas.

QoS
(Quality of Service). Garantías de calidad en el servicio.

RAD
(Rapid Application developtment). Desarrollo Rápido de Aplicaciones.

RDBMS
Sistema manejador de bases de datos relacionales. Es un DBMS que se basa en el sistema de bases de datos relacional.

Red
Grupo de computadoras interconectadas, incluyendo al hardware y al software usado para conectarlos.

Re-entrante
Se usa para describir código que puede tener invocaciones simultáneas, internivel o anidadas, que no interfieren unas con otras. Esto es importante para el procesamiento en paralelo, las funciones o subrutinas recursivas y el manejo de interrupciones.

Reuters
Compañía encargada de proporcionar información financiera oportuna a través de sistemas en línea.

Round-Robin
Algoritmo de calendarización, en el cual los procesos son activados en un orden cíclico fijo.

RMI
(Remote Method Invocation). Invocación a Métodos Remotos. Característica del lenguaje Java para desarrollar computación distribuida. Consiste en invocar al método de un objeto como si fuera local, pero en realidad está en otra parte de la red.

RPC
(Remote Procedure Call). Llamada a procedimientos remotos.

Sandbox
Es un ambiente protegido y delimitado, en donde las aplicaciones, como los programas Java descargados de Internet, se pueden ejecutar sin peligro de dañar al resto del sistema.

Scheduler
(Calendizador). Componente que tiene a su cargo la ejecución y administración de tareas relacionadas con el tiempo.

Script
Término aplicado a cualquier lenguaje que es frágilmente escrito y que contiene pocas o ninguna estructuras de datos. Generalmente es interpretado.

SDK
(Software Development Kit). Conjunto de Herramientas de Desarrollo de Software.

Servidor
Es la computadora que proporciona archivos o servicios.

Servidor de Aplicaciones
(Application Server). Es una aplicación en la capa intermedia que combina comunicación con sistemas back-ends; comunicación con clientes front-ends (a menudo, pero no necesariamente clientes Web); y un marco de trabajo en donde se puede colocar la lógica del negocio.

Servidor Web
Servidor que está conectado a Internet y es capáz de atender páginas Web.

Servlet
Es un programa Java que se ejecuta como parte de un servicio de red, generalmente es un servidor HTTP que responde a las solicitudes de clientes. Su uso más común es extender a un servidor Web, generando dinámicamente el contenido de las páginas.

Síncrono
Se dice de dos o más procesos que dependen de la ocurrencia de eventos específicos.

Sistemas Peer-to-Peer
Esquema de cooperación entre sistemas, en donde los componentes interactúan con idénticas jerarquías.

SMTP
(Simple Management Transport Protocol). Manejo Simple del Protocolo de Transporte.

SNA
(Systems Network Architecture). Arquitectura de Redes de Sistemas.

SQL
(Structured Query Language). Es un lenguaje y un estándar ANSI, desarrollado en los años 70 por IBM. Usa palabras regulares en Inglés para muchos de sus comandos. Frecuentemente está dentro de otros lenguajes de programación. Se utiliza para recuperar y actualizar la información de una base de datos.

SSL
(Secure Socket Layer). Protocolo de seguridad basado en llaves criptográficas.

Stateful
Se dice de un componente que puede permanecer activo entre invocaciones consecutivas a uno de sus métodos. Guarda el estado de las variables.

Stateless
Se dice de un componente que es desactivado después de cada llamada al método. No guarda el estado de las variables.

Stored procedure
Procedimiento Almacenado. Es un conjunto de comandos SQL y de control de flujos, almacenados bajo un nombre en un servidor.

Stub
En Jaguar, es una clase Java que actúa como un objeto proxy para un componente Jaguar. Compilado y ligado con una aplicación cliente en Java, un stub se comunica con Jaguar para instanciar e invocar a un método de un componente en la capa

intermedia. Los stubs hacen que los compoentes remotos de Jaguar parezcan locales para el cliente.

Sybase Central
Es una interfaz tipo Windows para administrar las bases de datos y productos relacionados.

SYBASE INC.
Proveedor de software enfocado al manejo de bases de datos.

TCP
(Transmission Control Protocol). Protocolo de Control de la Transmisión. Es utilizado en Internet. Usa el protocolo de Internet (IP) como protocolo subyacente.

TDS
(Tabular Data Stream) Flujo de datos tabular.

Thread-safe
Seguro de Hilos. Descripción de un código reentrante o protegido de ejecuciones múltiples simultáneas por alguna forma de exclusión mutua.

Time Slicing
Rebanadas de tiempo. Periodo de tiempo en el cual un proceso tiene permitido ejecutarse ininterrumpidamente en un sistema operativo multi-proceso.

Token Ring
Es un esquema de arbitraje en una LAN, en donde se evitan problemas en la transmisión de mensajes, mediante la garantía de los "tokens" o marcas que permiten el envío.

TP
Procesamiento de transacciones. Ejecución de las transacciones.

Transacción
Cualquier operación de actualización, inserción o borrado de información.

Trigger
Tipo de procedimiento almacenado en el servidor que se ejecuta automáticamente al ejecutar una transacción sobre una tabla.

Tunneling
Encapsulación de un protocolo dentro de otro. Se usa para obtener datos entre dominios administrativos que usan un protocolo que no es soportado por la Internet que conecta esos dominios.

TRS
(Transaction Router Service). Servicio de Ruteo de Transacciones. Programa de conexión directa que acepta solicitudes y las dirige a las conexiones abiertas del servidor.

UDP
(User Datagram Protocol). Protocolo de Datagrama de Usuario. Es un protocolo sin conexión que reside en la parte superior de IP.No garantiza la entrega ni si ésta necesita de una conexión. Es ligero y eficiente, pero todo el procesamiento de errores y de retransmisión deben ser cuidados por el programa de aplicación.

URL
(Uniform Resource Locator). Localizador de Recursos Uniforme. Forma estándar de especificar la localización de un objecto, generalmente una página Web en la Internet.

VSAM
(Virtual Sequential Access Method). Método de Acceso secuencial Virtual. Esquema de almacenamiento de archivos en disco creado por IBM.

WAN
(Wide Area Network). Red de Area Amplia. Red de comunicación de datos que sirve a usuarios a través de un área geográfica amplia y utiliza a menudo los dispositivos proporcionados por otras entidades, que se comparten entre varios clientes.

WebOLTP
(Net On Line Transaction Processing). Procesamiento de transacciones en línea y en red.

Wired
Alámbrico. Es un término que describe una red de computadoras conectadas físicamente.

Wireless
Inalámbrico. Es un término que describe una red de computadoras sin conexión física entre la que envía y la que recibe, sino que están conectadas por radio. Las aplicaciones para redes inalámbricas incluyen teleconferencias multi-partes, sesiones de trabajo distribuidas, asistentes digitales personales y periódicos electrónicos.

Wizard
Asistente. Es una utilería de ayuda interactiva que guía al usuario a través de una tarea compleja

Workflow
(Flujos de Trabajo). Integración de los procesos del negocio dentro de un sistema.

WWW

(Word Wide Web). Red Mundial. Conjunto de recursos desplegados en Internet que pueden ser accedidos desde un navegador.

XA

(Extended architecture). Arquitectura Extendida.

Bibliografía

Barlotta Michael. Taming Jaguar, Ed. Manning, USA, 2001.

Flanagan David. Java in a Nutshell, Ed. O'Reilly, 3.- edición, USA.

Goodman Danny. Java Script Bible, Ed. Mac Windows, 3.- edición, USA.

Jaguar
Manual de Jaguar CTS.

Sybase
Building Application Using Java and Enterprise Application Server
Student Guide, 1999.

Tenenbaum Andrew S. Redes de Ordenadores, Ed. Prentice-Hall, USA, 1991.

3- and n-Tier Architectures.
URL http://www.d-tec.ch/e/3tier.html
Fecha de Consulta : septiembre 2000.

Arquitectura de Negocios Multi-Plataforma de Múltiples Capas.
URL http://www.crear.com.ar/jbaspa.htm
Fecha de Consulta : Septiembre 2000.

Berkeley
URL http://www.sybase.com/detail/1,3693,210336,00.html
Fecha de Consulta : Enero 2001.

CORBA Tools and Utilities
URL http://www.ida.liu.se/~yuxzh/corba.html
Fecha de Consulta : Septiembre 2000.

Diccionario de Computación
URL http://foldoc.doc.ic.ac.uk/foldoc/index.html
Fecha de Consulta : Septiembre 2000.

Ericcson
URL http://www.sybase.com/detail/1,3693,1002962,00.html
Fecha de Consulta : Diciembre 2000.

Glosario
URL http://www.cis.upenn.edu/~cse330/lecture9/tsld026.htm
Fecha de Consulta : Septiembre 2000.

Introducing Jaguar CTS - The Component Transaction Server for WebOLTP.
URL http://my.sybase.com/detail?id=47622
Fecha de Consulta : Octubre 2000.

Jaguar CTS Programmer's Guide
URL
http://manuals.sybase.com/onlinebooks/group-jg/jgg0110e/jag_prog/@Generic__BookView/252;cs=default;ts=default
Fecha de Consulta : Noviembre 2000

Jaguar CTS (PowerSoft)
URL http://desarrollo1:8080/docs/index.html
Fecha de Consulta : Octubre 2000.

Jaguar CTS (Sybase)
Introducing Jaguar CTS - The Component Transaction Server for WEBOLPT
URL http://www.sybase.com/detail/1,3151,47622,00.html
Fecha de Consulta : Noviembre 2000.

Java
URL http://java.sun.com/a-z/index.html
Fecha de Consulta : Enero 2001.

Java
URL http://www.sun.com/java/platform.jhtml
Fecha de Consulta : Febrero 2001

Los Alamos
URL http://www.sybase.com/detail?id=210307
Fecha de Consulta : Diciembre 2000.

NetDynamics
URL http://www.netdynamics.com/
Fecha de Consulta: Agosto 2000

Northlandata
URL http://www.sybase.com/detail?id=210380
Fecha de Consulta : Diciembre 2000.

Object Web
URL
http://www.sims.berkeley.edu/courses/is206/f97/GroupF/objectweb.html#Introduction
Fecha de Consulta : Octubre 2000.

Overview Project
URL http://pbserv.baruch.cuny.edu/instructor/overview.htm
Fecha de Consulta : Octubre 2000.

Servidores de Aplicaciones
URL
http://dir.yahoo.com/Business_and_Economy/Business_to_Business/Communications_and_Networking/Internet_and_World_Wide_Web/Software/World_Wide_Web/Servers/Application_Servers/
Fecha de Consulta: Agosto 2000

Sybase
Taking it to the Web
URL http://www.sybase.com/detail/1,3151,1002583,00.html
Fecha de Consulta : Diciembre 2000.

WebLogic
URL http://edocs.bea.com/index.html
Fecha de Consulta: Agosto 2000

WebSphere
URL http://www-4.ibm.com/software/webservers/
Fecha de Consulta: Agosto 2000