ESCUELA SUPERIOR POLITECNICA DEL LITORAL

FACULTAD DE INGENIERIA EN ELECTRICIDAD Y COMPUTACION

TRABAJO DE GRADUACIÓN

"APLICACIÓN BANCARIA EN SU CASA"

Previo a la obtención del Título de:

INGENIERO EN COMPUTACIÓN

Presentado por:

Paúl Gavilanes Balarezo
Alex Montesdeoca Hidalgo
Manuel Vega Ulloa

GUAYAQUIL – ECUADOR
1999

A G R A D E C I M I E N T O

Al ING. CARLOS VALERO

Director de Tesis y Profesor de los tópicos de graduación, por su ayuda y colaboración para la realización de este trabajo.

D E D I C A T O R I A

A NUESTROS PADRES

A NUESTROS HERMANOS

A NUESTRAS ESPOSAS

A NUESTROS HIJOS

TRIBUNAL

.....

ING. CARLOS VALERO

.....

ING. GUIDO CAICEDO

.....

ING. REBECA ESTRADA

DECLARACIÓN EXPRESA.

La responsabilidad por los hechos , ideas y doctrinas expuestos en esta tesis, nos corresponden exclusivamente; y, el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL.

(Reglamento de Exámenes y Títulos profesionales de la ESPOL)

Paúl Gavilanes B. Alex Montesdeoca H. Manuel Vega U.

RESUMEN.

La Aplicación Banca en Casa fue desarrollada para proveer servicios bancarios similares a los que los clientes encontrarían en los bancos, pero en la comodidad de su propio hogar.

Esta aplicación fue diseñada usando la arquitectura de Aplicaciones Cliente / Servidor usando CORBA (Common Object Request Broker Architecture), en la cual se usó a DB2 como Servidor de la base de datos, el WebSphere Application Server como servidor Web y el Visual Age for Java como lenguaje de programación de la aplicación en sí. Para conectarse a la aplicación es necesario un código de usuario y su respectiva clave de acceso a través de un Web browser (navegador) conectado a Internet.

Para el desarrollo de nuestra aplicación hemos considerado los siguientes requerimientos, los cuales los hemos divido en dos: los requerimientos de las opciones que debe brindar la aplicación al cliente, y los requerimientos de la aplicación en cuanto a los productos o herramientas sobre la cual funciona.

Requerimientos del Cliente**:**

Los clientes que usen la Aplicación Bancaria deben estar en posibilidad de:

- Accesar a sus saldos de cuentas corrientes y de ahorros.
- Accesar las historias de los movimientos de sus cuentas.
- Transferir fondos entre sus cuentas.
- Crear cuentas.
- Transacciones de débitos y retiros de fondos de sus cuentas.
- Cambiar su clave de acceso o datos personales.

Los clientes deben poder hacer estas transacciones en forma segura y desde cualquier computadora conectada a Internet. También decidimos que los clientes deberían autentificarse por segunda vez cuando se haga una transacción que efectúe cambios en los saldos de las cuentas.

Requerimientos de la aplicación:

Los requerimientos iniciales, basados en los requerimientos de la aplicación son:
Un cliente de la aplicación necesita:

- Un código de usuario.
- Una clave de acceso.
- Al menos una cuenta activa en el banco.
- Un browser de Internet.

- Acceso a Internet.

Un proveedor de los servicios implementados por la Aplicación Bancaria necesita:

- Un servidor de aplicación.
- Un servidor Web.
- Un servidor de servlets.
- Soporte para CORBA.

Esta aplicación fue desarrollada usando una arquitectura de tres capas:

La primera capa es el browser HTML, el cual muestra las páginas HTML enviadas por el Web Server, y también envía la información necesaria al Web Server para procesar las transacciones a través de los applets.

La segunda capa es el Web Server y el Application Server. El Web Server coordina, colecta y ensambla las páginas web con contenido estático o dinámico y las envía al cliente. El cliente se comunica con el Web Server por medio del protocolo estándar conocido como HTTP. El Application Server controla del lado del servidor todos los procesos java. Estos procesos son responsables de manejar la lógica del negocio.

La tercera capa contiene la base de datos y la capacidad transaccional. Los recursos de esta capa son accesados a través de la capa intermedia. La base de datos es responsable de almacenar toda la información necesaria para la aplicación. La explicación de la arquitectura del sistema solo concierne los datos que involucra la aplicación. Si nos concentramos en el origen, procesamiento y almacenamiento de la información esto es suficiente. Por ejemplo en el login, el usuario y el password se originan en la primera capa, son pasados a la segunda capa, algo de procesamiento ocurre (esto involucra a la base de datos), y una respuesta es enviada al browser de la primera capa.

La Aplicación Bancaria en Casa sigue una arquitectura Modelo / Vista / Controlador, cuyo propósito es separar la lógica de la presentación (la Vista) de la lógica del negocio (el Modelo). Esto se logra usando un Controlador, el cual actúa como un puente a través del cual se coordina la actividad entre la vista y el modelo.

Uno de los propósitos de la arquitectura Modelo / Vista / Controlador es ayudar a dividir el desarrollo de la aplicación en distintos roles. En un escenario de desarrollo Web algunos roles pueden ser especialistas gráficos, programadores java, y programadores de la lógica del negocio. El punto aquí es definir responsabilidades con una mínima interferencia con otros roles.

INDICE GENERAL

Parte I: Introducción

Capítulo 1: Introducción.

Esta aplicación provee servicios de Home Banking (Banca en el Hogar) a los usuarios a través de un Web browser haciendo uso de todos los beneficios y servicios que nos provee actualmente la gran red mundial de información internet.

1.1 Acerca de la Aplicación Home Banking.

Las compañías están siempre tratando de obtener una ventaja competitiva sobre sus rivales. Esto ocasiona que se esfuercen en hacer mucho más fácil que la gente en todo el mundo haga negocios con ellas. Las tiendas de libros han encontrado que vender a través del World Wide Web incrementó notablemente los volúmenes de ventas a través del mundo. Esta es una oportunidad que pocos negocios pueden ofrecer.

La industria de la banca no es la excepción. Las instituciones financieras tradicionalmente han sido inaccesibles y poco cooperativas. Los clientes típicamente debían hacer sus transacciones en persona en la agencia más cercana al lugar de su residencia y esperar inclusive una cantidad indeterminada de horas. Todo esto ha cambiado con el internet, ahora muchos bancos grandes ofrecen un conjunto completo de servicios a sus clientes en internet. Hay un claro beneficio en esto tanto para los bancos como para sus clientes. Los clientes pueden realizar sus transacciones desde donde quieran y cuando quieran. Los bancos tienen la posibilidad de atraer clientes desde lugares muy lejanos. La Aplicación de Banca en Casa (ABC) provee de toda la funcionalidad que deberían tener las aplicaciones comerciales para la banca.

1.2 Funcionalidad Básica.

La Aplicación de Banca en Casa (ABC) permite al cliente ejecutar transacciones básicas de banca en el internet usando un web browser. Los siguientes servicios son ofrecidos por el ABC:

• Consulta de Saldos de Cuentas Corrientes y de Ahorros.

• Movimiento de Cuentas Corrientes y de Ahorro.

• Transferencias de Fondos.

Para la utilización de estos servicios se navega a través del web browser en el sitio indicado por el banco (http://hostname). Se presenta una pantalla principal de la ABC, desde donde se pide el usuario y el password asignados. Una vez verificado el usuario se procede a escoger los servicios que ofrece la ABC.

1. Inicio - Información inicial del sitio.

2. Cuentas – Permite el acceso a la información de las cuentas

3. Transferencias – Transfiere dinero entre cuentas personales

4. Transacciones – Transacciones entre cuentas personales

5. Logout – Terminar la sesión.

Si los usuarios están interesados en revisar el saldo de sus cuentas, deben hacer click en "Cuentas". Se mostrará una lista con las cuentas activas y sus respectivos saldos. Desde aquí se podrán revisar los movimientos de cualquier cuenta haciendo click en "Detalle".

1.3 Arquitectura del Sistema.

Se pueden utilizar diferentes tecnologías para desarrollar la ABC, pero nos concentraremos en una arquitectura de tres capas:

La primera capa es el browser HTML. El browser muestra las páginas HTML enviadas por el Web Server. El browser también envía la información necesaria al Web Server para procesar las transacciones a través de los applets.

La segunda capa es el Web Server y el Applicaction Server. El Web Server coordina, colecta y ensambla las páginas web con contenido estático o dinámico y las envía al cliente. El

cliente se comunica con el Web Server por medio del protocolo estándar conocido como HTTP. El Application Server controla del lado del servidor todos los procesos java. Estos procesos son responsables de manejar la lógica del negocio.

La tercera capa contiene la base de datos y la capacidad transaccional. Los recursos de esta capa son accesados a través de la capa intermedia. La base de datos debería ser responsable de almacenar toda la información necesaria para la aplicación. Desde el punto de vista del cliente se debería incluir opciones como las de usuarios y passwords. Típicamente, estos sistemas son muy avanzados y la misión crítica de ellos es integrar a la organización con el Web. La explicación de la arquitectura del sistema solo concierne los datos que involucra la aplicación. Si nos concentramos en el origen, procesamiento y almacenamiento de la información esto es suficiente. Por ejemplo en el login, el usuario y el password se originan en la primera capa, son pasados a la segunda capa, algo de procesamiento ocurre (esto involucra a la base de datos), y una respuesta es enviada al browser de la primera capa.

La Aplicación de Banca en Casa sigue una arquitectura Modelo / Vista / Controlador. El propósito de esta arquitectura es separar la lógica de la presentación (la Vista) de la lógica del negocio (el Modelo). Esto se logra usando un Controlador, el cual actúa como un puente a través del cual se coordina la actividad entre la vista y el modelo.

En el caso de la ABC, la parte visual de la aplicación debería ser mostrada en el Web browser del cliente. El controlador debería estar formado por los java servlets que toman los mensajes desde el cliente. El modelo debería ser cualquier servicio relacionado a la lógica del negocio específicamente (como acceso a la base de datos por ejemplo).

Uno de los propósitos de la arquitectura Modelo / Vista / Controlador es ayudar a dividir el desarrollo de la aplicación en distintos roles. En un escenario de desarrollo Web algunos roles pueden ser especialistas gráficos, programadores java, y programadores de la lógica del negocio. El punto aquí es definir responsabilidades con una mínima interferencia con otros roles.

1.4 Tecnologías y Productos.

En esta tópico nos hemos enfocado en la arquitectura de objetos distribuidos más madura existente al momento, Common Object Request Broker Architecture (CORBA).

Las especificaciones de CORBA guian la interoperabilidad de los objetos desarrollados con cualquier lenguaje de programación. Java es el lenguaje más escogido para crear estos objetos usando CORBA. Los Enterprise Java Beans (EJB) son el resultado final de usar Java y CORBA.

Como hemos visto el usar Java y CORBA son la mejor manera de construír aplicaciones cliente / servidor actualmente, el siguiente paso es determinar qué productos les proveen soporte.

Debido a que la mayor parte del desarrollo será hecho en Java, nuestra herramienta favorita para crear los applets de Java, las aplicaciones, los Objetos CORBA, los Java Beans, y los EJB, es Visual Age para Java.

Para entornos CORBA, tenemos 2 alternativas: IBM WebSphere y Component Broker. Desde la perspectiva CORBA, el uso de WebSphere es una implementación ligera de CORBA, y la Component Broker es una implementación que direcciona las aplicaciones de misión crítica en la empresa. Para el desarrollo del presente proyecto se seleccionó Websphere y los componentes CORBA implementados por IBM para este servidor de Aplicaciones.

Otras tecnologías fueron utilizadas para el desarrollo e implementación del proyecto, como las diferentes herramientas de programación de páginas web provistas en el mercado.

1.4.1 Visual Age y WebSphere.

Implementamos la Aplicación Bancaria usando objetos CORBA ejecutándose en el entorno WebSphere.

Para desarrollar la aplicación, se usó el entorno de desarrollo IDL (Interface Definition Language). Visual Age, nos ayudó a crear y manejar stubs y skeletons a partir de archivos IDL utilizando compiladores externos.

1.4.2 WebSphere y CORBA.

En el desarrollo de nuestra aplicación tenemos 2 partes fundamentales: el lado del cliente y el lado del servidor. Para ambos lados, Websphere provee de una implementación de las especificaciones CORBA que puede ser fácilmente importado en el Visual Age, haciendo más rápido el desarrollo de aplicaciones.

1.4.3 Integración.

El paso final en el desarrollo de la aplicación es la integración de la misma. El objeto ORB se ejecuta en una máquina diferente y es accesado desde el primer servidor ejecutando el servidor de HyperText Transfer Protocol (HTTP), el servlet de WebSphere y el entorno CORBA.

Nuestra versión final del sistema integra IIOP y clientes HTML puros, y del lado del servidor objetos CORBA WebSphere, servlets creando páginas HTML dinámicas.

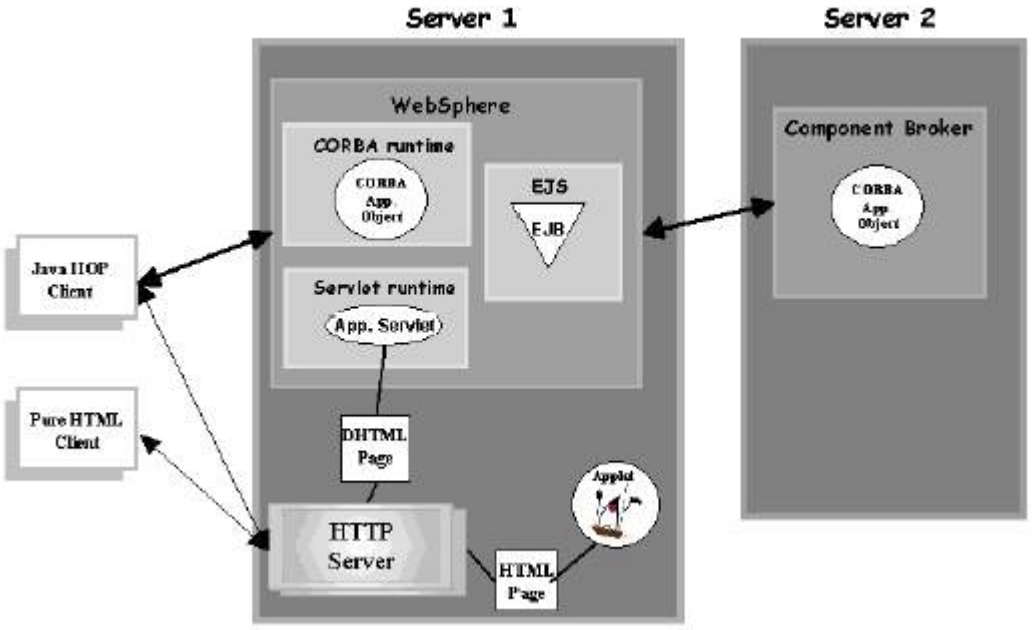


Figura 1.4: esquema de la Aplicación Bancaria

Parte II: Fundamentos Teóricos

Capítulo 2: Objetos Distribuidos.

En los últimos 30 años se han visto algunos cambios en la manera de diseñar, desarrollar y mantener los sistemas de información de las empresas. Hoy en día predominan las redes de computadoras. En el futuro veremos aplicaciones distribuídas a través de redes heterogéneas, computadoras, y sistemas operativos usando diferentes lenguajes, tecnologías y almacenamiento de datos.

La separación de la interfaz ("lo que es") de su implementación ("como es") será como se alcance aplicaciones verdaderas que cumplan con diversas plataformas, diversos lenguajes y diversos diseños.

2.1 Aplicaciones Monolíticas.

En las aplicaciones monolíticas, llamadas también 1-tier (1 capa), todos los accesos a los datos y las lógicas de negocios y de presentación, se encuentran en la misma máquina. No es posible compartir datos comunes, pero las aplicaciones deberían compartir datos y poder comunicarse con otras aplicaciones, así que tendríamos que tener los mismos datos replicados en todas las máquinas. También deberíamos tener las mismas funciones de acceso a los datos a través de las aplicaciones. Esto causa problemas de sincronización y desempeño.

Para este tipo de aplicaciones deberíamos tener máquinas muy poderosas, y cuando se sobrecarguen, la única solución sería la actualización de las mismas. Las aplicaciones monolíticas son costosas y difíciles de mantener.

2.2 Aplicaciones Cliente / Servidor.

Para mejorar las aplicaciones 1-tier (1 capa), se desarrollaron las aplicaciones 2-tier (2 capas), llamadas también aplicaciones cliente / servidor. Esta arquitectura divide una

aplicación monolítica en 2 componentes monolíticos localizados en diferentes máquinas. La máquina servidor provee acceso al sistema de manejo de la base de datos relacional (RDBMS) e implementa algo de la lógica del negocio.

La máquina cliente implementa la lógica de la presentación y la lógica del negocio. Por lo tanto ambos lados pueden compartir datos en común, pero el cliente es un cliente pesado debido a que tiene que implementar la lógica de la presentación y algo de la lógica del negocio.

La figura 2.2 nos muestra la arquitectura de una aplicación cliente / servidor:

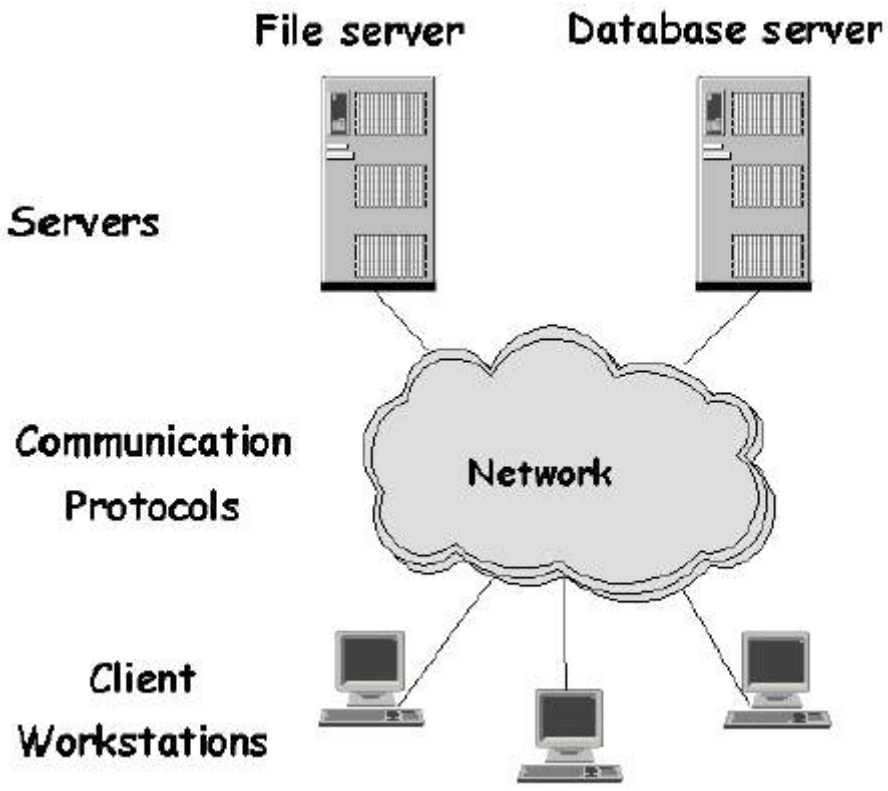


Figura 2.2: Arquitectura Cliente / Servidor.

2.3 Aplicaciones Distribuídas.

Con la introducción del desarrollo de aplicaciones orientadas a objetos, podemos diseñar y desarrollar aplicaciones flexibles e interoperables, y el proceso de desarrollo se hace menos costoso y en menor tiempo que antes.

Los objetos pueden ser distribuídos a través de las diferentes redes y sistemas operativos, y cada componente puede interoperar con los demás componentes sin importar el sistema operativo ni el tipo de red física usados. Lo cual nos trae ciertos beneficios, tales como infraestruturas flexibles, portabilidad, reuso de código, e interoperabilidad.

Debemos considerar las tecnologías y sistemas antiguos, no podemos desecharlos, ya que se invirtió demasiado tiempo y dinero en el diseño, desarrollo, mantenimiento y pruebas de los mismos.

El desarrollo orientado a objetos nos permite integrar las tecnologías antiguas con las nuevas usando encapsulamiento. Por ejemplo, podemos tener un módulo escrito en un leguaje procedural como un objeto, y reusarlo sin problemas. Usando encapsulamiento para la integración y reingeniería es posible usar la funcionalidad del software ocultando el código detrás de un nuevo modelo de objeto.

Debido a que el repositorio de los objetos se encuentra en el servidor, la aplicación cliente puede mantenerse ligera y pequeña, lo cual nos da ventaja para el mantenimiento de la misma. No tenemos que redistribuir todo nuestro código modificado debido a que tenemos los objetos en el servidor. El cliente usa una interfaz de dicho objeto, así cuando se modifica la implementación del objeto, su interfaz no se afecta.

Otra ventaja de la programación orientada a objetos es la habilidad de reusar código. Los objetos distribuídos pueden ser divididos en módulos separados que pueden ser modificados independientemente. Cada módulo o componente puede comunicarse con los demás componentes, por lo tanto no es necesario reinventar la rueda. Los desarrolladores pueden trabajar simultáneamente, y cuando hayan terminado los nuevos objetos, pueden construír la aplicación usando estos módulos y reusando los existentes.

Otra ventaja de los objetos distribuídos se relaciona con la distribución de código. Con la mayor parte de la aplicación distribuída en los servidores, la distribución de código al cliente

se lo maneja intercambiando los componentes modificados del lado del servidor, por lo tanto, se elimina la necesidad de actualizaciones masivas a todos los clientes. La figura 2.3 nos muestra un entorno de arquitectura 3-tier (3 capas).

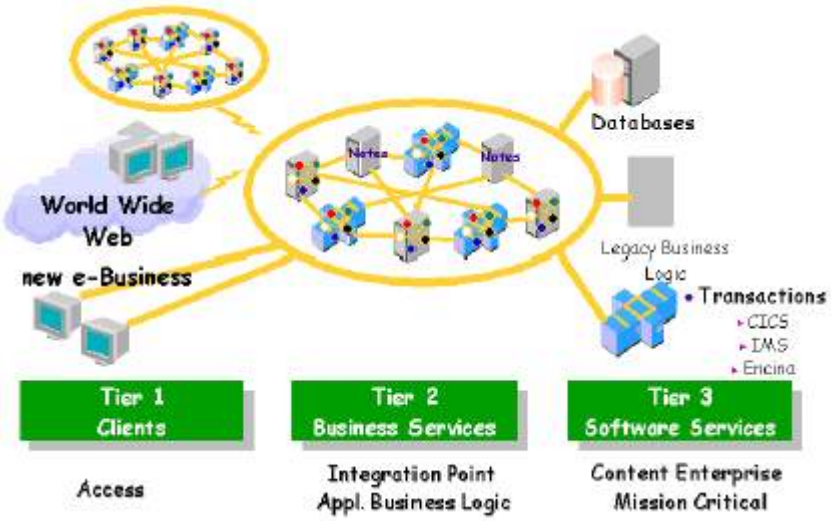


Figura 2.3: Entorno multicapa.

Capítulo 3: CORBA.

Existen muchas razones para elegir usar una infraestructura de objetos distribuídos para una aplicación empresarial.

Las tareas más comunes que un sistema distribuido realiza son:

- Crear, localizar, y compartir objetos.
- Realizar queries a objetos.
- Autentificación y validación.
- Administrar el acceso compartido a los objetos a través de bloqueo, deadlocks, condiciones de acceso, afinamiento del desempeño.
- Servir a múltiples clientes en un ambiente multihilos.
- Proveer cache en el servidor o en cliente.
- Administración de la aplicación (incluye escalabilidad, uso eficiente de los recursos existentes, balanceo de carga, y agregar nuevos servidores).

3.1 Object Management Group (OMG).

La OMG fue establecida en 1989 como un consorcio abierto de más de 800 compañías que trabajan juntas para definir estándares abiertos para la arquitectura de objetos distribuídos. Algunos de estos estándares son CORBA, IIOP, Servicios de Objetos, Facilidades de Internet, y especificaciones para Interfaz de Dominio. La primera publicación de la OMG fue hecha en 1990.

CORBA 1.1, introducido en 1991, especificaba el IDL que define la interacción entre objetos del cliente y del servidor en una implementación específica de un ORB.

CORBA 2.0, adoptada en Diciembre de 1994, introdujo verdadera interoperabilidad entre ORBs de diferentes vendedores.

3.2. Componentes de una implementación CORBA.

Una implementación CORBA tiene 4 elementos:

- El ORB, a través del cual los objetos se intercomunican.

- Servicios de Objeto, que definen los servicios a nivel de sistema que son agregados al ORB. Ejemplos de estos servicios son: nombramiento, seguridad, persistencia, transacción.

- Facilidades comunes, las cuales definen servicios a nivel de aplicación. Ejemplos de estos servicios son: componentes, documentos compuestos, entre otras facilidades.

- Objetos de aplicación, las cuales capturan el comportamiento real del entorno, tales como cuentas de bancos, clientes, y aviones.

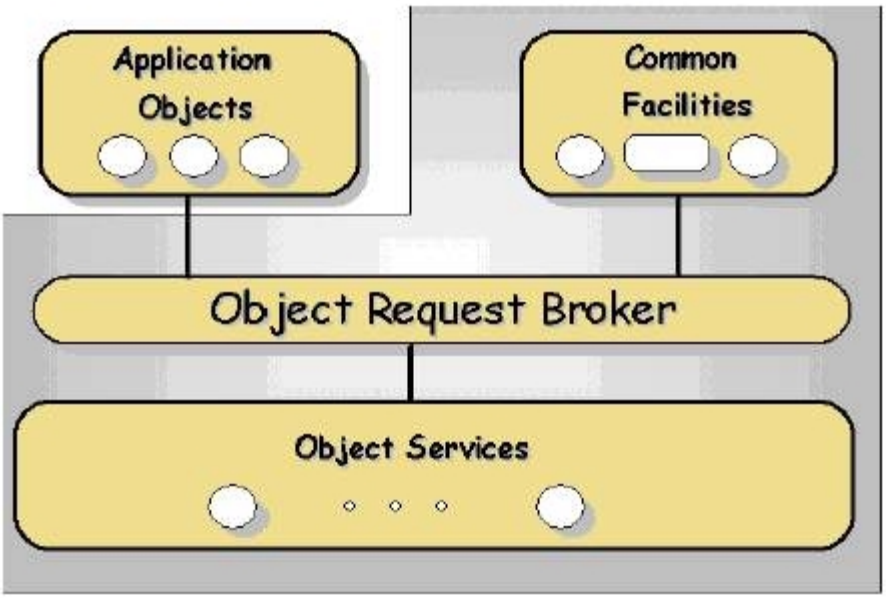


Figura 3.2: relaciones entre los elementos de CORBA.

3.2.1 Object Request Broker (ORB).

El ORB es la pieza de middleware que establece la relación cliente / servidor entre los objetos. Intercepta cualquier requerimiento que el cliente pide, es responsable de encontrar un objeto que pueda implementar el requerimiento, pasarle los parámetros requeridos, invocar el método, y retornar el resultado al cliente. El cliente no necesita conocer la localización física del objeto, ni el lenguaje de programación, ni el sistema operativo, ni cualquier otra especificación sobre el objeto, el ORB se encarga de estos detalles.

El ORB es el único componente de CORBA que debe estar presente siempre para que CORBA pueda funcionar como lo hace. Algunos vendedores ofrecen implementaciones sin los servicios o las facilidades de CORBA. Usted debe crear estos componentes o comprarlos por separado. El ORB, sin embargo, debe estar presente para que su aplicación distribuida funcione.

Algunas de las responsabilidades del ORB son:

- Localizar e instanciar los objetos en máquinas remotas.

- Administrar los parámetros de manera consistente entre lenguajes y sistemas operativos.

- Invocar métodos en objetos remotos, usando invocación estática (en tiempo de compilación) o invocación dinámica (en tiempo de ejecución).

- El uso de métodos de callback desde el servidor a los objetos apropiados en los clientes.

El ORB provee transparentemente todos estos servicios y más. Como desarrollador, todo lo que tiene que hacerse es proveer los parámetros apropiados de inicialización del ORB, y el resto lo maneja la implementación del ORB.

3.2.2 Servicios de Objeto.

La OMG ha publicado especificaciones para 15 diferentes servicios de objetos, los cuales veremos brevemente a continuación:

- Ciclo de vida: define las operaciones de creación, copia, movimiento, y eliminación de objetos.

- Persistencia: provee una interfaz para almacenar objetos de manera persistente en los servidores, que incluye RDBMS, sistemas de manejo de objetos bases de datos, y archivos planos.

- Nombramiento: permite la localización de objetos por el nombre. El nombre es convertido a una referencia de objeto interoperable (IOR) lo que se conoce como name binding. Este proceso también permite que los objetos sean asociados a otros protocolos tales como X.500, DCE, NIS+, Netware Directory Services (NDS).

- Eventos: permite a los objetos registrar un interés específico en eventos que pueden ser disparados por otros objetos. El servicio de eventos maneja la comunicación entre los eventos consumidores y los productores. En otras palabras, los productores proveen datos al servicio de eventos, el cual los pasa a los eventos consumidores.

- Control de concurrencia: coordina el acceso de múltiples clientes a los mismos recursos. Está diseñado principalmente para ser usado en ambientes transaccionales donde se pueden establecer bloqueos para el correcto fin de una transacción.

- Transacción: provee coordinación para las transacciones que se originan de diferentes clientes en un sistema distribuido. También es conocido como Object Transaction Service (OTS). EL OTS puede empezar una unidad de trabajo, coordinar el trabajo hecho por cada participante, y realizar commit de dos fases (two-phase commit).

- Relaciones: permite relaciones entre los objetos para ser definidos explícitamente. Además, Reglas de tipos y de cardinalidad pueden expresarse.

- Externalización: define una convención para lectura y escritura de un objeto serializado, desde y hacia una fuente de datos, por ejemplo, la escritura de un objeto en un proceso fuera de línea. Este servicio está muy alineado con el Servicio de Relaciones.

- Query: permite a los usuarios y a los objetos realizar queries en otros objetos. Le permite usar indíces, y se basa en un estándar de SQL (Structured Query Language) y OQL (Object Query Language).

- Licenciamiento: provee un mecanismo para los productores de controlar el uso de la propiedad intelectual. Algunos ejemplos son: controlar el número de usuarios concurrentes, periodos de tiempo de evaluación, y esquemas de retroalimentación.

- Propiedades: provee operaciones que agregan valor (propiedades) al objeto. Estas propiedades son en adición a los atributos del IDL estático definido.

- Tiempo: provee una interfaz para sincronizar el tiempo sobre la red distribuida. El servicio de tiempo es a menudo usado para determinar el orden absoluto de los eventos entre objetos distribuidos.

- Seguridad: provee autentificación, autorización, log de seguridad, y administración de seguridad.

- Trader: provee un servicio de comparación entre objetos. Registra la disponibilidad de un servicio y los objetos puede preguntar por los objetos que reúnen ciertas condiciones.

- Colección: soporta un conjunto de comportamientos para un número común de colecciones, tales como conjuntos, colas, pilas, listas, y árboles.

3.2.3 Facilidades comunes.

Son colecciones de servicios que son usados nivel de aplicación para mejorar la consistencia del desarrollo distribuido. Una facilidad común define la manera en la cual los componentes del negocio interactúan entre sí.

Las facilidades comunes que están en construcción incluyen agentes móviles, flujo de trabajo, frameworks de objetos de negocios, e internacionalización. La OMG revisa constantemente cuales facilidades son más necesarias y por lo tanto también la IDL para la cual se producen cambios regularmente.

3.2.4 Objetos de aplicación.

Estos objetos encapsulan el comportamiento del negocio. Cada objeto de aplicación conoce como responder a los requerimientos, lanzar eventos, y manejar relaciones con otros objetos de aplicación. El término aplicación es muy usado, un objeto de aplicación es cualquier objeto que encapsula y provee un conjunto de comportamientos discretos.

Para que los clientes interactúen con los objetos de aplicación, estos objetos deben cumplir ciertas reglas de acoplamiento, las cuales son definidas por CORBA, y especifican como un objeto de aplicación se hace conocer al resto del mundo y de cuales servicios dispone para los clientes.

Algunas compañías tratan de crear un conjunto de frameworks que direccionen la lógica genérica del negocio. Un par de estos frameworks son San Francisco de IBM, y BOF (Business Object Framework) de la OMG. En cada framework hay algunos frameworks de dominio específico, tales como manufactura, cuidado de la salud, telecomunicaciones, transporte, finanzas, pero cada uno se basa en el objeto raíz.

Algunas aplicaciones distribuidas no necesitan del overhead de la infraestructura compleja y definen en su lugar un conjunto de objetos específicos del negocio que se basan en el objeto raíz.

3.3 Internet Inter-ORB Protocol (IIOP).

En CORBA 1.1, la OMG no especificó el protocolo que los ORBs usarían para comunicarse entre sí. Como resultado, los vendedores produjeron ORBs que usaban capas de comunicaciones y protocolos disparejos.

Con CORBA 2.0, la OMG introdujo un nuevo protocolo de comunicación llamada IIOP. Para que un ORB cumpla con las especificaciones de CORBA, debe usar IIOP como protocolo de comunicación para interactuar con los ORBs. IIOP es una implementación específica de TCP/IP del GIOP.

EL GIOP especifica formatos de mensajes y representación común de los datos (CDR). Como resultado, las implementaciones de los ORBs pueden comunicarse entre ambientes heterogéneos. La CDR se preocupa de las diferencias entre las diversas plataformas, tales como la representación de punto flotante entre otras cosas.

El GIOP define el formato para los IORs. Un ORB usa un IOR como identificador único, y por consiguiente para localizar un objeto específico.

Capítulo 4: Visual Age para Java Empresarial.

Visual Age for Java fue diseñado para el desarrollo completo de aplicaciones. Usando sus herramientas, se puede ganar productividad en comparación a otras herramientas de su misma especie.

4.1 Desarrollo Dinámico e Iteractivo.

El ciclo tradicional de desarrollo editar-compilar-depurar puede tomar mucho tiempo en ambientes competitivos actuales. Rompiendo la secuencia natural de este ciclo, Visual Age for java realiza el desarrollo completo iteractivo.

4.1.1 Control de Fuentes Simplificado.

Con Visual Age for Java, no debemos preocuparnos por el manejo de archivos y respaldos frecuentes de los programas fuentes. Cuando un elemento del programa es creado, automáticamente es almacenado en el repositorio. Una vez creado, el elemento del programa estará también disponible en el ambiente de desarrollo integrado (Integrated Development Environment) o workspace. Los elementos de programas creados con anterioridad, pueden ser añadidos al workspace conforme sean necesarios. Mientras el workspace contiene solo una edición simple del elemento del programa (llamada edición actual), el repositorio contiene todas las ediciones. Cuando el elemento del programa es añadido al workspace, el IDE compila el programa fuente en Java bytecodes. Similarmente, cuando un elemento del programa es creado, la primera vez que es grabado, es colocado en el repositorio y la compilación es automática.

El IDE también provee de una poderosa herramienta de comparación para los objetos del repositorio. Esta herramienta es especialmente usada para comparar diferentes ediciones del mismo objeto. Cuando las ediciones de la misma clase son comparadas, cualquier diferencia es mostrada, en diferentes modos: fuente cambiado, añadido o eliminado.

4.1.2 Ejecución Interactiva.

Cuando un ambiente de desarrollo que permite a los programadores ingresar e inmediatamente ejecutar el código, sin necesidad de procesos batch o links, se dice que soporta ejecución interactiva.

La ejecución interactiva permite a un programa ser desarrollado y probado incrementalmente desde abajo. El programador crea nuevo código y prueba su interactividad sin tener herramientas especiales para estas pruebas. La productividad de los usuarios de lenguajes compilados se ve afectada por el ciclo editar-compilar-depurar. En contraste con los usuarios de lenguajes interpretados, que simplemente ingresan y ejecutan las sentencias en un considerable tiempo.

Java fue diseñado para ser interpretado, pero su tipo de sistema permite ser compilado de manera efectiva.

Visual Age for Java emula múltiples instancias de la máquina virtual. Cuando una instancia de una clase es ejecutada, a través de su método main o como un applet, una nueva instancia de la máquina virtual es creada para esta, y cada instancia de la máquina virtual trabaja por separado.

4.1.3 Compilación y Enlace Incremental

Los lenguajes compilados como C++ requieren de un esquema físico de objetos conocido en tiempo de compilación. Esto permite que sea generado código eficiente, pero cuando una definición de una clase es cambiada, cualquier clase que la usa tiene que ser recompilada. Si una clase de bajo nivel es cambiada, virtualmente la aplicación por entero tiene que ser recompilada. El tiempo de compilación y enlace para aplicaciones complejas puede llevar varios minutos y hasta horas. Este tiempo puede afectar seriamente la productividad del desarrollo porque el efecto de un pequeño cambio al código fuente no puede ser rápidamente evaluado.

Java fue diseñado para evitar las recompilaciones innecesarias. Los bytecodes de Java no contienen información acerca del esquema físico de otras clases de Java que las use. En cambio, un archivo Java bytecode contiene referencias simbólicas a los métodos y campos de otras clases. Estas referencias simbólicas son resueltas en tiempo de ejecución, cuando la clase es cargada para su ejecución. De modo que, un cambio a cualquier clase no afecta a ninguna clase que la usa.

Visual Age for Java toma la inherente capacidad incremental de Java para su conclusión lógica: en lugar de usar un archivo basado en un modelo de compilación, Visual Age for Java compila el código fuente incrementalmente cuando un método o clase es editada y guardada. El compilador chequea la sintaxis del código fuente y construye una lista de métodos y campos de las cuales depende. Cuando estos elementos del programa son cambiados, el compilador determina el impacto y marca cualquier método invalido. El Solucionador de Problemas lista todos los problemas del código, así el desarrollador puede navegar el área del problema y corregirlo.

Recordemos que Visual Age for Java emula múltiples instancias de la máquina virtual. Es así, como el código puede ser cambiado durante la ejecución. Visual Age for Java es un poderoso ambiente de desarrollo de programas de larga ejecución, tales como servidores.

4.1.4 Depuración.

El soporte de la depuración es una función crítica para cualquier ambiente de desarrollo. Existen tres categorías de tecnologías de depuración, cada una de las cuales tiene su uso práctico:

Depuración Invasiva.
Insertando sentencias de impresión en el código y directamente la salida a la consola o a archivos log.

Depuración Remota.

Adjuntando un depurador a otro, posiblemente remoto, proceso JVM y seguimiento paso a paso a través del código.

Depuración Local.

Depurando múltiples instancias de la máquina virtual ejecutándose dentro del IDE, y modificando el código durante la ejecución.

Visual Age for Java soporta depuración local y remota. De cualquier manera, las aplicaciones Java que utilizan varias máquinas virtuales y que normalmente requerirían depuración remota, pueden ejecutarse dentro de Visual Age for Java debido a su capacidad de emular múltiples instancias de la máquina virtual.

Se pueden seleccionar hilos de ejecución individuales para la revisión. Cuando un hilo es seleccionado, su pila de llamada es mostrada. Se puede seleccionar cualquier pila y ver el código fuente para este método y la lista de variables. Mientras el código es mostrado, puede ser editado y grabado, esto causa que el método sea automáticamente compilado y enlazado dentro del programa en ejecución.

Consideramos que esta característica es muy importante en el caso de depurar servlets. Supongamos que un servlet es desarrollado en un servidor Web que usa una JVM estándar, y que, después de muchos requerimientos, un problema ocurre. Debido a que la máquina virtual estándar no se carga incrementalmente una clase en el servidor Web debería detenerse, ser inicializada y probada nuevamente para recrear las condiciones bajo las cuales ocurrió el problema, esto cada vez que el código sufra cambios. Todo estos inconvenientes y otros más, son soportados por Visual Age for Java.

4.2 Paradigma de Construcción por Partes.

El modelamiento de componentes y programación visual se complementan entre sí. Las partes creadas en base a un modelo de componentes son manipulados en ambiente visual.

4.2.1 Programación Visual.

Visual Age for Java ofrece un ambiente de programación visual que es significativamente diferente de otras herramientas de desarrollo. Las herramientas de programación permiten definir el esquema visual de los applets o aplicaciones con una simple selección y mecanismos desde una paleta. Algunas herramientas asisten en la generación de código para eventos particulares y los relaciona a los controles. Esta asistencia típicamente toma la forma de un experto que genera el código. Hay una completa separación de la composición o esquema visual de la interfaz del usuario y la lógica que maneja su conducta, que es, el soporte que existe para la integración de las partes no visuales sobre una superficie visual.

El Composition Editor de Visual Age for java nos ofrece no solo los elementos de la interfaz del usuario, sino que también permite visualizar la conducta del applet o aplicación. Usando conexiones visibles entre los elementos visuales y los no visuales, se puede definir e integrar la lógica de la interfaz. No sólo se pueden mostrar o manipular los componentes no visuales, sino también se puede añadir beans con elementos no visuales en los canvas. Estos beans tienen una representación visual y puede ser manipulados tal como los otros componentes visuales.

4.2.2 Soporte a JavaBeans.

Con el modelo de componentes JavaBeans, se pueden construir aplicaciones enteras con partes reusables, aumentando la potencial efectividad del ciclo de desarrollo. Lo que separa a los JavaBeans de los otros componentes es su independencia de la plataforma usada.

Visual Age for Java soporta una variedad de maneras de generar y añadir código a una aplicación o bean. A través del Bean Info Page, se trabaja con código desde la perspectiva de un bean. Las propiedades, los métodos, y los eventos pueden ser añadidos, y además las propiedades pueden ser designadas como índices o fronteras. Si se define manualmente un evento, se puede apreciar todo el código que se genera a través de esta herramienta.

El estándar para JavaBeans distribuidos en un paquete con todas las clases relacionadas y los archivos de recursos, es un archivo jar. Visual Age for Java soporta la importación y exportación de archivos jar desde el IDE.

4.3 Características adicionales.

Luego de una revisión del Visual Age for Java podemos listar otras características importantes:

- Soporte de programación en equipo. Visual Age for Java Empresarial provee un ambiente de desarrollo en equipo integrado que se basa en un repositorio de código fuente compartido.

- *Herramientas empresariales.* Estas herramientas añaden escalabilidad a sus aplicaciones Java e incluyen herramientas poderosas para en desarrollo en Java. Los compiladores del alto desempeño, maximizan la velocidad de ejecución en su código servidor. Desde una estación de trabajo, se puede depurar probar y ejecutar herramientas de análisis para aplicaciones con soporte para OS/2, Windows NT, AIX, OS/400, OS/390 y AS/400.

- *Mapeo de relaciones entre objetos.* El Constructor de Persistencia automatiza el mapeo del estado de persistencia de los objetos Java a Base de Datos Relacionales.. Esto genera una capa con código que implementa todas las llamadas de acceso JDBC necesarias para insertar, actualizar o recuperar datos para un objeto desde una base de datos SQL.

- *Aplicaciones Corba en Java.* El Ambiente de Desarrollo IDL puede ser usado para desarrollar aplicaciones con objetos distribuidos que corren en IIOP y ambientes Component Broker o en ambientes tres-partes tales como VisiBroker y Orbix.

- *Constructor de Sevlets*. Con Visual Age for Java Empresarial, se pueden usar técnicas de programación visual para crear y probar servlets. Usando el Constructor de Servlets y a través del Servidor de Aplicación IBM WebSphere, se pueden probar y depurar sitios Web construidos con HTML puro, Java Server Pages compiladas (JSP) y servlets visuales.

- *Constructor de Accesos Empresariales.* Esta herramienta permite crear aplicaciones Java que usan aplicaciones y datos de host. Incluye conectores CICS y Encina; estos beans son parte del mismo Common Connector Framework (CCF) que usa los conectores IBM e-Business.

- *Generadores Expertos.* Para generar componentes de aplicaciones.

- *El Asistente de Migración.* El cual transforma componentes ActiveX en JavaBeans.

Capítulo 5: IBM Websphere Aplication Server y DB2 Universal Database.

5.1 ¿ Qué es el IBM WebSphere Aplication Server ?

El IBM WebSphere Application Server hace uso de la frase “escríbalo una vez, y uselo en cualquier parte” para el desarrollo con servlets. Este producto consiste de una máquina servlet basada en Java que es independiente de su Web server y su sistema operativo. El Application Server le ofrece plug-in de servidores compatibles con las más populares interfaces de programación en aplicaciones tipo servidor (APIs). Los Web server que soporta son:

- IBM HTTP Server
- Apache Server
- Domino
- Lotus Domino Go Webserver
- Netscape Enterprise Server
- Microsoft Internet Information Server

5.2 Características del WebSphere Application Server.

Adicionalmente a la máquina servlet y los plug-ins, el Application Server provee:

- Implementación de la librería API java Servlet de JavaSoft, más algunas extensiones.
- Aplicaciones de ejemplos para demostrar las clases bases y sus extensiones.
- El WebSphere Application Server Manager, una interfaz gráfica en donde se puede realizar fácilmente:

- ✓ Cambio de opciones para cargar servlets locales o remotos.
- ✓ Cambio de parámetros de inicialización.
- ✓ Manejo de servlets.
- ✓ Especificar alias a los servlets.
- ✓ Crear cadenas de servlets y filtros.
- ✓ Administrar y monitorear los componentes de los Enterprise Java Services.
- ✓ Log de mensajes para los servlets.
- ✓ Habilitar la depuración en la JVM
- ✓ Monitorear los recursos usados por el Application Server
- ✓ Monitorear los servlets cargados, las sesiones activas y las conexiones JDBC
- ✓ Monitorear los errores, los eventos, y las excepciones.
- ✓ Habilitar y deshabilitar dinámicamente el seguimiento.

- Un manejador de conexiones que utiliza cache y reusa las conexiones a su JDBC accesando a las bases de datos. Cuando un servlet necesita una conexión a una base de datos, puede obtenerla de un pool de conexiones disponibles, eliminando el overhead requerido para abrir una nueva conexión para ese requerimiento.

- Clases adicionales en Java para el manejo de los JavaBeans, que permiten a los programadores el acceso a la bases de datos por JDBC. Estos Beans de acceso a datos proveen funcionalidad, mientras esconden la complejidad del uso de base de datos relacionales. Pueden ser usados de manera visual en un ambiente de desarrollo integrado.

- Soporta contenidos de páginas dinámicas conocido como JavaServer Pages (JSP). La tecnología JSP produce páginas Web dinámicas con un lenguaje de scripts del lado del servidor. El objetivo es separar la lógica de presentación (por ejemplo el código que define la estructura del sitio Web y su apariencia) de la lógica del negocio (por ejemplo, el código java que accesa a base de datos para mostrar información en el Web). Por flexibilidad, los archivos JSP pueden incluir los servlets y los JavaBeans.

- Enterprise Java Services (EJS), ejecutan y manejan las aplicaciones basadas en especificaciones de Enterprise JavaBeans de Sun.

5.3 Los Componentes IBM WebSphere Application Server.

Los siguientes componentes forman el Application Server:

- Una máquina basada en servlets.
- Una máquina basada en Java.
- Web server plug-ins.
- Enterprise Java Services (EJS).
- Application Server Manager.
- Una interfaz gráfica fácil de usar para configurar, manejar y monitorear
- JavaServer Pages (JSP).
- Ejemplos.
- Documentación.

5.4 IBM DB2 Universal Database.

DB2 Universal Database de IBM es un sistema de administración de bases de datos relacionales que son el punto más crítico de muchos sistemas de negocios. Esta base de datos es compatible con aplicaciones Web, escalable desde simples procesadores hasta multiprocesadores simétricos. También tiene características multimedia con imágenes, audio, video, texto y otras ventajas. Con las nuevas versiones se continúa la evolución de la tecnología de bases de datos en aplicaciones Web como soporte adicional Java, funciones cliente / servidor, etc.

5.5 Características de DB2 Universal Database.

Compatibilidad.

Gracias a su alcance mundial y relativo bajo costo, el internet se ha convertido en una herramienta comercial de desarrollo. Pero si realmente se quiere tomar ventaja de esta oportunidad, se tiene que estar listo para mantener un sitio web las 24 horas del día los 7 días de la semana. Se necesita construir páginas rápidas de cargar y fáciles de navegar, así también, se desea permitir a los usuarios completar transacciones seguras y exitosas.

Cuando se añade DB2 a los proyectos de aplicaciones Web, se pueden satisfacer todas estas necesidades y más. En efecto, con las siguientes capacidades DB2 esta listo para el Web:

Soporte Java, con JDBC puede accesar a una variedad de plataformas de clientes. Usando JDBC puede permitir que el cliente accese a DB2 a través de applets Java desde cualquier Web browser que permitan Java. Alternativamente se pueden construir aplicaciones servidores completamente en Java y el acceso a DB2 se puede realizar a través de JDBC. Adicionalmente DB2 soporta SQLJ que permite crear, construir y ejecutar sentencias estáticas SQL para aplicaciones java tales como procedimientos almacenados.

Otras características que demuestran su compatibilidad son:

El *Control Center,* el cual está *basado en Java*, esto permite manejar una base stand alone o multiusuarios desde un centro de control en un Web browser

Provee *seguridad, con autentificación y autorización de servicios*, que pueden ser integrados fácilmente con servicios de redes y sistemas operativos.

Diseñada para entregar rápido acceso, a bases de datos dinámicas que es lo que se necesita como soporte de los negocios en el Web.

Soporte de objetos multimedia, DB2 permite entregar una completa experiencia de los usuarios del Web.

Escalabilidad, herramientas de alto desempeño como el IBM WebSphere Application Server ponen sus aplicaciones Web basadas en DB2 en un ambiente competitivo.

Herramientas de conectividad de datos, son una puerta de enlace a sus datos corporativos. Los datos que las aplicaciones tienen resultan en un alto desempeño y escalabilidad al explorar los avanzados tipos de datos en las aplicaciones DB2.

Distribución.

Al momento de escoger una solución de base de datos para cada aplicación se tiene que observar la capacidad de procesar transacciones y distribuir sofisticadas aplicaciones inteligentes. Se tienen pocas opciones para lograr esto, pero DB2 Universal Database lo hace mejor. DB2 nos da el poder de procesamiento paralelo para soportar las aplicaciones de negocios inteligentes con consultas complejas o base de datos de gran tamaño. Esto se debe sumar al soporte seguro de un ambiente de cientos de usuarios en un alto número de transacciones. DB2 reescribe las consultas automáticamente para optimizar el resultado de la misma.

Universabilidad.

Más del 70 por ciento de las mayores compañías del mundo confían a DB2 el manejo de sus aplicaciones críticas del negocio. Con DB2 se puede desarrollar toda aplicación que requiera alguna de las siguientes características:

- Procesamiento de Transacciones en línea (OLTP).
- Data Warehousing.
- Soporte de toma de decisiones.
- Data Mining.

Existen más de 4,000 desarrolladores de software independientes incluyendo SAP, Baan and PeopleSoft que ofrecen diversas aplicaciones que soportan DB2.

Multimedia.

Debido a que DB2 soporta objetos multimedia, se puede almacenar texto, imágenes, videos, audio y otros tipos de datos objetos. Se puede buscar en la base de datos casi cualquier criterio que se pueda imaginar. DB2 es la primera base de datos que soporta multimedia gracias a sus DB2 Extenders.

Disponibilidad.

Para muchas organizaciones la disponibilidad 24x7 es usual en sus negocios. DB2 hace esto más fácil asegurándose que sus aplicaciones estén ejecutándose y ofreciendo a sus usuarios las siguientes características:

- Soporte en variedad de ambientes que permite definir una base de datos que se ejecute en múltiples servidores.
- Respaldo en línea disponible 24x7; respaldo fuera de línea ofrece máxima eficiencia.
- Respaldos a nivel de tablas y recuperación de los mismos.
- El Control Center también ayuda a la estrategia de replicación.

Consultas paralelas.

DB2 Universal Database incorpora la capacidad de consultas paralelas, con lo cual ha llamado la atención de muchas organizaciones que ejecutan consultas sobre las bases de datos de gran tamaño y necesitan una manera rápida de obtener respuestas a sus preguntas del negocio.

DB2 Universal Database ayuda a la consulta dividiendo una simple consulta del usuario en partes pequeñas, las cuales son ejecutadas simultáneamente a través de sistemas de procesamiento paralelo. Se reescribe la consulta si es necesario para optimizar su ejecución,

luego, automáticamente crea un plan de acceso para la ejecución en paralelo. La búsqueda de datos, los enlaces, los ordenamientos, la carga de datos, la creación de índices, los respaldos, y las recuperaciones son ejecutadas simultáneamente.

Facilidad de Manejo

Los expertos de las industrias y los usuarios son favorecidos con las nuevas herramientas de administración en DB2 Universal Database. El Control Center incluye una interfaz a través del Web browser para accesar y manipular los objetos de la base de datos, una tarea calendarizada, logs, etc. Así mismo, cuenta con el SmartGuide, que es un conjunto de ventanas de diálogo a través de las cuales, paso a paso, se optimiza la configuración de su base de datos.

Capítulo 6: Java.

En lugar de describir el lenguaje java, detallaremos algunos atributos principales del lenguaje respecto al entorno de programación. También abarcaremos los Enterprise Java Beans y sus implicaciones respecto a CORBA.

6.1 Atributos de Java.

Veremos los atributos que hacen de Java una excelente elección para el desarrollo de aplicaciones distribuidas.

6.1.1 Orientado a objetos.

Java es un lenguaje orientado a objetos. Por lo tanto como desarrolladores, podemos concentrarnos en los datos y en el comportamiento del negocio, en lugar de tener que pensar en término de los procesos. La naturaleza de un lenguaje orientado a objetos permite el reuso de código, en distintos niveles de proyecto y de empresa.

Sun ha provisto una amplia librería de clases para aumentar la funcionalidad del JDK. Algunas de estas características para las cuales Java ha definido especificaciones son acceso a las bases de datos, seguridad, telefonía, gráficos en 3 dimensiones, reconocimiento de voz, correo, imágenes, y ayudas. Con este gran conjunto de funciones puede tomar menos tiempo el desarrollo de código que ya ha sido escrito por otros vendedores.

6.1.2 Arquitectura neutral.

Java tiene una arquitectura (o plataforma) neutral usando los llamados bytecodes. Estos bytecodes no son una tecnología nueva, debido a esto, la JVM (Java Virtual Machine) ha alcanzado gran soporte de la industria muy rápidamente. La JVM es una porción de software escrita para cada sistema operativo específico. Interpreta los bytecodes y realiza el comportamiento deseado para ese sistema operativo.

Consideremos el caso donde los bytecodes le dicen a la JVM que muestre una ventana. El resultado final depende de la implementación de la JVM. Por ejemplo, la JVM de Windows 95 muestra la ventana de acuerdo al estilo de Windows 95, la JVM de AIX muestra la ventana de acuerdo al estilo de AIX. Es así como si aparecen nuevos sistemas operativos, los vendedores pueden escribir una JVM de acuerdo a su sistema operativo y por lo tanto permitir que los programas en Java se ejecuten en sus plataformas sin hacer cambios en los bytecodes.

6.1.3 Multihilo.

Tradicionalmente, el desarrollo multihilo ha sido un arte para pocos desarrolladores. Sin embargo, Java hace la propiedad rmultihilo más fácil debido a que incluye soporte para multihilos como parte del lenguaje. Existen clases para multihilos que los objetos pueden heredar y la palabra reservada del lenguaje synchronized, que permite al desarrollador proveer de acceso de semáforos a las variables, métodos, o clases.

Cuando se desarrolla una aplicación GUI, es fácil caer en la trampa de tener un solo hilo, es decir, si el usuario empieza una acción que lleva hacerla algunos segundos (o minutos), el usuario se bloquea de otras tareas hasta que la primera tarea concluya. Sin embargo, Java hace fácil el hacer nuevas tareas a través de hilos, por lo tanto libera al usuario para que pueda ejecutar más de una tarea en paralelo.

Una aplicación en el servidor debe ser capaz de servir múltiples requerimientos de múltiples clientes en paralelo. El uso de hilos, y la manera en que Java los provee, hace que escribir una aplicación para el servidor sea una opción más factible.

6.1.4 Dinamismo.

Cualquier clase de Java puede ser cargada en la JVM en cualquier momento. Las instancias de estas clases pueden ser dinámicamente cargadas y luego instanciadas. En cualquier

momento puede ejecutarse un query a la clase objeto para determinar sus atributos y comportamiento usando una técnica llamada reflejo. Reflejo le habilita a usted para que dinámicamente pueda invocar un método de una instancia, construír una instancia de un objeto, usar un string que contenga el nombre de una clase, y alterar valores de una instancia dada.

6.1.5 Seguridad.

Java tiene algunos controles para asegurar un entorno seguro. Los punteros no pueden ser usados para accesar memoria directamente. Un verificador de bytecodes se ejecuta en cada clase cuando es cargada, para asegurar que los bytecodes estén de acuerdo a un estricto conjunto de reglas. Algunas de las cosas que el verificador revisa son si los bytecodes son válidos y que en la pila no se producirá overflow o underflow.

Java también tiene lo que se conoce como sandbox, la cual se asemeja a una caja de arena de un niño, la sandbox de Java limita la cantidad de travesuras que el programa Java pueda cometer. La sandbox de Java implementada en los Web browsers, previene que el applet sea leído desde el disco duro, escribir en el disco, conectarse a otras computadoras, y otras actividades peligrosas. Java usa certificados digitales para permitir el acceso a la computadora del usuario. Un certificado digital garantiza que el proveedor de los applets sea quien dice que es. Asumiendo que el usuario confía en el proveedor del applet y acepta el certificado, las restricciones de la sandbox pueden ser parcial o totalmente levantadas.

6.1.6 Portabilidad.

Esta es una de las más grandes fortalezas de Java. Los desarrolladores pueden escribir código en Java, ya que saben que podrá ser compilado y ejecutado en cualquier plataforma que soporte Java (NT, OS/2, AIX, Solaris, HP, Linux, MVS, AS/400, DEC, entre otras).

Sun ha anunciado una iniciativa "100 % Java Puro" para hacer que los desarrolladores se aseguren y certifiquen que sus códigos son portables. Técnicamente es posible escribir código no portable en Java, sin embargo, Sun y otros están tratando de que los desarrolladores "Escriban Una Vez y Ejecuten en Cualquier Lugar".

La portabilidad es un reto en un ambiente de ORBs, hace que sea un poco más fácil el manejo de la interoprabilidad entre ORBs. A pesar de que siempre habrá problemas con la comunicaciones disparejas entre ORBs, tener el resto del lenguaje consistente entre ellos es un excelente punto de partida.

6.2 Applets.

Las características básicas de los applets son:

- Se ejecutan dentro de un Web browser en la máquina del cliente.
- Son bajados desde la máquina servidor.
- Tiene acceso limitado en la máquina del cliente.
- Tienen acceso limitado a la red, solo pueden accesar al servidor desde el cual proviene.
- La mayor ventaja de los applets es son automáticamente distribuidos a la máquina del cliente. No hay mantenimientos y las nuevas versiones del applet sólo son instaladas en el servidor
- El Web browser provee la JVM.

Capítulo 7: Programación Web.

Este capítulo introduce las tecnologías para desarrollo de aplicaciones e-business y Web con las cuales estamos familiarizados hasta el momento. En este momento ya tenemos conocimientos sobre HTTP, browsers, servidores Web, así como también del lenguaje de programación Java.

7.1 Modelo de programación Web.

Una aplicación Web es cualquier aplicación que usa tecnologías Web, incluyendo los Web browsers, los servidores Web y los protocolos de Internet.. Las aplicaciones Web típicamente se conectan a otros servidores tales como sistemas de bases de datos o de transacciones (Figura 7.1).

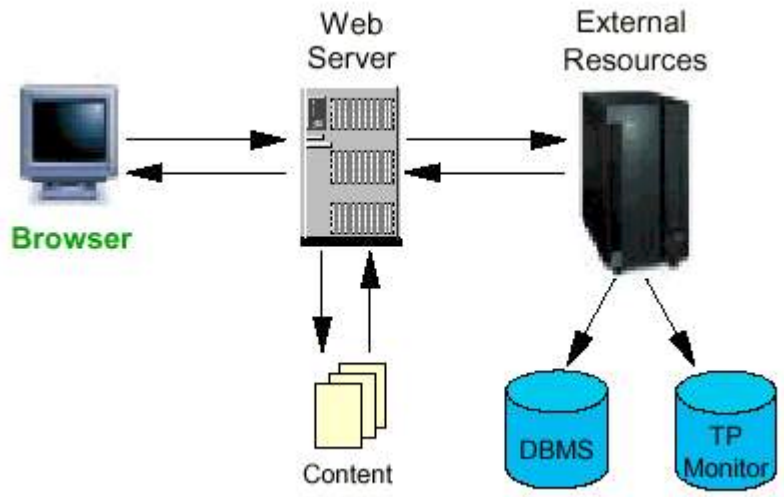


Figura 7.1: Componentes de una aplicación Web.

El modelo de programación Web usa una arquitectura multicapa, lo cual significa que las aplicaciones son particionadas en componentes. La primera capa es el browser del Web. La segunda capa incluye el servidor Web (y un servidor opcional de aplicación) que construye

las páginas Web estáticas o dinámicas y las envía a los clientes. En nuestra aplicación, la lógica de la segunda capa es implementada en Java usando servlets y JavaServer Pages.

La tercera capa provee servicios tales como los de la base de datos y los de transacciones. Típicamente, estos son sistemas maduros de negocios los cuales las organizaciones desean integrarlos en el Web.

7.2 Generación de página dinámica.

Las páginas dinámicas son usadas para proveer a la aplicaciones Web salida hacia los Web browsers. Estas páginas son enviadas por el servidor a petición del cliente, por ejemplo, para consultar los precios de una bodega o de fábrica en el Web, las páginas dinámicas requieren hacer más que enviar el contenido de una página estática HTML al browser.

Las principales tecnologías que soportan páginas dinámicas son:

CGI (Common Gateway Interface).

En este modelo un nuevo proceso es creado para cada requerimiento desde el browser. A pesar de que es muy simple de implementar y ser soportado por la mayoría de los servidores Web, tiene un pobre desempeño debido a que un nuevo proceso tiene que ser lanzado para cada requerimiento HTTP que accesa el programa CGI, limitando el número de requerimientos concurrentes que el servidor puede manejar. Adicionalmente, un programa CGI no puede interactuar con el servidor Web mientras se encuentra en ejecución debido a que se ejecuta en un proceso por separado.

Lenguajes Script.

Algunas compañías han creado del lado del servidor entornos de scripts, incluyendo entre otras a NetData de IBM, JavaServer Pages de Microsoft. Estas tecnologías son muy populares y permiten a los constructores de sitios Webs incluir contenido dinámico como scripts directamente en las páginas Web. El lado malo de estas tecnologías es que todavía están limitadas a un grupo particular de productos o de sistemas operativos y admás el desarrollador debe aprender los lenguajes de scripts.

Tecnologías de plug-in en el servidor.

Existen muchas de estas tecnologías soportadas por muchos servidores Web. Estas tecnologías proveen muy buen desempeño pero dependen mucho del servidor Web y son muy complicadas de programar. Estas tecnologías incluyen NSAPI de Netscape e ISAPI de Microsoft.

Servlets.

Son la solución Java al contenido dinámico, y son explicados en detalle en la sección "Servlets" más adelante.

JavaServer Pages (JSP).

Es una nueva tecnología que consiste en páginas con código precompilado que se ejecuta obteniendo valores de propiedades o métodos de objetos del lado del servidor.

7.3 Servlets.

Los Servlets son programas del lado del servidor hechos en Java que se ejecutan en un servidor Web o de aplicaciones. Los servlets son para el servidor lo mismo que los applets

son para los browsers. Los servlets son cargados y ejecutados en un servidor Web y los applets son cargados y ejecutados en un browser.

Los servlets son definidos por la API de los Java servlets, la cual define una interfaz estándar entre un servlet y un servidor. Esto los hace portables a través de los servidores.

7.3.1 Características de los servlets.

Entre las principales características de los servlets tenemos:

Portabilidad.

Los servlets son escritos en Java haciéndolos portables a través de las plataformas y de muchos servidores Web debido a que el API de los servlets define una interfaz estándar entre un servlet y el servidor Web.

Persistencia y Desempeño.

Un servlet es cargado una sola vez por el servidor Web, e invocado por cada requerimiento del cliente. Esto significa que el servlet puede mantener los recursos del sistema (como la conexión a una base de datos) entre requerimientos, y no hay overhead al instanciar un nuevo servlet por cada requerimiento. Los servlets pueden ser cargados dinámicamente o cuando el servidor Web es inicializado.

Basados en Java.

Debido a que los servlets son escritos en Java, heredan todos los beneficios de Java, incluyendo lo fuerte del sistema, la orientación a objetos, y la modularidad. Debido a la existencia del garbage collection y de la ausencia de manipulación de punteros, los servlets ayudan a evitar problemas en el manejo de la memoria que pueden tener otras aplicaciones.

7.3.2 Acceso a los Servlets.

Los servlets son accesados desde un browser de las siguientes maneras posibles:

- *Formas HTML.* Los servlets son por lo general invocados por el botón de submit en las formas HTML. Los datos ingresados por el usuario son pasados al servlet usando los métodos POST y GET.

- *Enlaces de hipertexto.* Los servlets pueden ser accesados por enlaces de hipertexto al igual que cualquier URL. El servicio o el método doGet del servlet es invocado. Los servlets pueden también ser invocados usando otros requerimientos tales como PUT y DELETE.

- *Etiqueta servlet.* Algunos servidores Web soportan la etiqueta HTML SERVLET o soportan servlets si el lado del servidor incluye la sintáxis <|-- include -- >. El servicio o método doGet es invocado y la salida es puesta en la página HTML, reemplazando la etiqueta SERVLET.

- Otros servlets: los servlets pueden accesar otros servlets ya cargados usando: *GetServletContext().getServlet("nombre sevlet");*

7.3.3 API de un servlet.

Define una interfaz estándar entre el servidor Web y el servlet. Los requerimientos del cliente son hechos al servidor Web el cual invoca al servlet para atender los requerimientos a través de la interfaz. La API está compuesta de dos paquetes:

javax.servlet

javax.servlet.http

El paquete javax.servlet contiene las clases para dar soporte a los servlets genéricos independientes de los protocolos. Esto quiere decir que los servlets pueden ser usados para cualquier protocolo que soporte el paradigma requerimiento / respuesta.

Similar a un applet, un servlet no tiene un método main, tiene un conjunto de métodos, o puntos de entrada, los cuales son invocados por el servidor. Un servlet es creado de una clase Java implementando la interfaz del servlet. Típicamente esto se hace instanciando ya sea la clase GenericServlet, para servlets independientes del protocolo, o la clase HttpServlet para servlets HTTP.

En este proyecto se trabajamos con el JSDK (Java Servlet Development Kit) el cual implementa el Servlet API 2.0, implementado en el WebSphere Application Server 2.02 de IBM.

7.3.4 Ciclo de Vida de un Servlet

El cliente de una aplicación basada en Servlets usualmente no se comunica directamente con un servlet, pero si le hace un requerimiento a través del servidor Web el cual invoca al servlet a través del API del servlet. El rol del servlet es inicializar, invocar el método service, y destruir cada instancia del servlet. Por lo general hay una instancia de cada servlet, con múltiples hilos creados para manejar múltiples requerimientos de los clientes (Figura 7.3.4). Esta característica hace que los servlets sean muy eficientes.

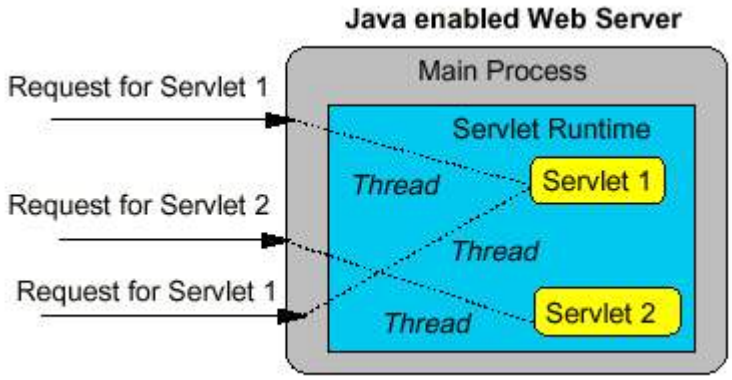


Figura 7.3.4: Modelo de ejecución de un Servlet.

Los servlets pueden ser cargados dinámicamente cuando sus servicios son requeridos por primera vez, o el servidor Web puede ser configurado para que ciertos servlets sean cargados cuando se inicializa al servidor.

Luego de que han sido cargados, el servidor Web se comunica con un servlet a través de la interfaz del Servlet la cual define cinco métodos: init, service, destroy, getServletConfig, y getServletInfo.

Init.

Este método es llamado cuando el servlet es cargado por primera vez. Una subclase de GenericServlet o HttpServlet sólo necesita implementar este método si debe realizar tareas de configuración que deban ser realizadas una vez en lugar de que sea durante cada requerimiento de los clientes. Un ejemplo de esto es la inicialización de una conexión a una base de datos o el cargar datos por omisión. El método init es llamado una sola vez, y es completado antes de que cualquier requerimiento sea manejado.

Service.

Cada vez que un requerimiento de un cliente es hecho, este método es llamado y son pasados los objetos ServletResponse y ServletRequest. Este método es responsable de construir la respuesta para el requerimiento del cliente.

Una subclase de HttpServlet no implementa este método. Cuando el servidor llama al método service de HttpServlet, determina si el requerimiento es GET o POST, y llama el método doGet o doPost apropiado para los cuales el desarrollador del servlet debe proveer la implementación apropiada.

DoPost.

Es invocado cuando un requerimiento HTTP POST es enviado a través de una forma HTML. Los parámetros asociados con un requerimiento POST son comunicados desde el browser al servidor como un requerimiento HTTP por separado. Un método doPost debe ser usado cuando se necesiten hacer modificaciones en el servidor.

DoGet.

Es invocado en respuesta a un método HTTP GET enviado por una URL o una forma HTML. Un método HTTP GET es el default cuando se especifica una URL en un browser. Al contrario del método doPost, doGet debe ser usado cuando no se necesiten hacer modificaciones en el servidor. Los parámetros asociados con un requerimiento GET son anexados a la URL y pasados en el requerimiento HTTP.

La respuesta del servlet puede ser de diversos tipos:

- Una cadena de salida, la cual el browser interpreta basado en el tipo de contenido, por lo general una página HTML.
- Una respuesta HTTP de error.

- Una redirección a otra URL usando el método sendRedirect.
- Una invocación de una JavaServer Page usando el método callPage.

Destroy.

Este método es llamado cuando el servidor Web descarga el servlet. Una subclase de GenericServlet o HttpServlet sólo necesita implementar este método si necesita realizar operaciones de limpieza, tales como soltar una conexión a una base de datos o cerrar archivos.

GetServletConfig

Este método retorna una instancia de ServletConfig, la cual puede ser usada para retornar los parámetros de inicialización y el ServletContext. La interfaz del ServletContext provee información sobre el entorno del servlet, y acceso al log del mismo.

GetServletInfo.

Este método es opcional, el cual provee información sobre el servlet, tales como su autor, la versión, y los derechos de autor.

Los métodos service, doGet, y doPost son invocados con un objeto de requerimiento y de respuesta el cual provee información sobre el requerimiento y las formas de comunicar la respuesta al browser. Estas clases son: javax.servlet.ServletResponse, javax.serletServletRequest para GenericServlets; javax.servlet.http.HttpServletRequest y javax.servlet.http.HttpServletResponse para HttpServlets.

7.4 Java Server Pages (JSP).

Es una tecnología del lado del servidor que permite la generación dinámica de la respuesta en el servidor. Usando JSP, se puede poner lenguaje de scripts embebido en una página HTML y accesar la lógica del negocio a través de scriptlets o de Java Beans.

Un servlet tradicional usa una cadena de salida para escribir código HTML para el servidor Web para mostrarlo en un browser. Los programadores que escriben el código de los servlets no son por lo general los diseñadores y puede que no produzcan páginas Web de una buena apariencia. Usando Java Server Pages se pueden separar las tareas de programación de los servlets de las tareas de diseño de las páginas HTML.

7.4.1 Especificaciones de las Java Server Pages.

Son una nueva tecnología. En el momento de escribir la versión 1.0 de las especificaciones, estaba siendo revisada la versión actual 0.92, sin embargo, pocas implementaciones existen. Por ejemplo, el WebSphere Application Server de IBM soporta una versión modificada de la versión 0.91 de JSP y será actualizado a la versión 1.0 o posterior en el futuro.

7.4.2 Características de las Java Server Pages.

Entre otras características tenemos las siguientes:

Separación del contenido de la presentación y la generación.

La responsabilidad del contenido y los datos es delegada a los componentes en el lado del servidor, mientras que las JavaServer Pages se encargan de extraer el contenido y mezclarlo con el documento HTML.

Mejor Arquitectura Modelo/Vista/Controlador.

Proveen un mejor soporte para la arquitectura Modelo/Vista/Controlador en una aplicación Web que los servlets. Antes de las JavaServer Pages, los servlets eran responsables de la lógica del control y la generación del contenido dinámico, lo cual hace que el mantenimiento de la aplicación sea más difícil.

Separación de los roles en el equipo de desarrollo.

Teniendo la lógica del negocio encapsulada en componentes, la lógica del control manejada por los servlets, y los HTML dinámicos manejados por las Java Server Pages, hace que sea más fácil identificar roles en el equipo de desarrollo.

Debido a que son archivos separados las JavaServer Pages, pueden ser mantenidos por un autor de HTML con un programador responsable por los Servlets y los Java Beans. El autor de los HTML puede interactuar con los Java Beans y los Servlets a través de etiquetas, añadiendo etiquetas al documento HTML.

Portabilidad y Familiaridad.

Si se usa Java como lenguaje script, los Java Beans como la arquitectura de componentes, y estándares como HTML para las presentaciones, las Java Server Pages son muy portables a través de las plataformas y los servidores Web. Si se usa Java como el modelo de programación y HTML para la presentación, las Java Server Pages se montan sobre conjuntos de habilidades existentes.

Basados en Java.

Debido a que las Java Server Pages se basan en Java, heredan todos los beneficios de Java incluyendo lo poderoso del sistema, la orientación a objetos, la modularidad y el buen manejo de la memoria.

7.4.3 API de una Java Server Page.

Esta API define la comunicación entre el código Java (usualmente servlets u otras Java Server Pages) y una Java Server Page. Existen 2 tipos:

Com.sun.server.http.HttpServiceResponse: extiende sun.sevlet.http.HttpRequest y provee un nuevo método: callPage para invocar una JavaServer Page.

Com.sun.server.http.HttpRequest: extiende sun.servlet.http.HttpResponse y provee un nuevo método setAttribute, para setear atributos del objeto request. Estos atributos pueden ser accesados en la Java Server Page usando la etiqueta BEAN.

También puede ser pasada la información a la JavaServer Page usando el método putValue del objeto httpSession para asociar objetos en la sesión. Estos objetos son accesibles mientras la sesión esté activa y los objetos seteados en el requerimiento, son sólo accesibles durante el tiempo de vida del requerimiento.

7.4.4 Cómo funcionan las Java Server Pages.

La primera vez que una Java Server Page es invocada (o cuando es cambiada), es llevada a un archivo fuente que contiene el servlet, luego es compilada e inicializada. Luego que el servlet es inicializado, el método service es invocado. Para todos los envios subsecuentes, el método service del servlet existente es invocado y la salida del servlet, una combinación de elementos estáticos y dinámicos (creados a través de elementos JSP), es enviada al browser

tal como muestra la Figura 7.4.4. Una Java Server Page tiene una extensión .jsp para que sea identificada por el servidor como archivo JSP.

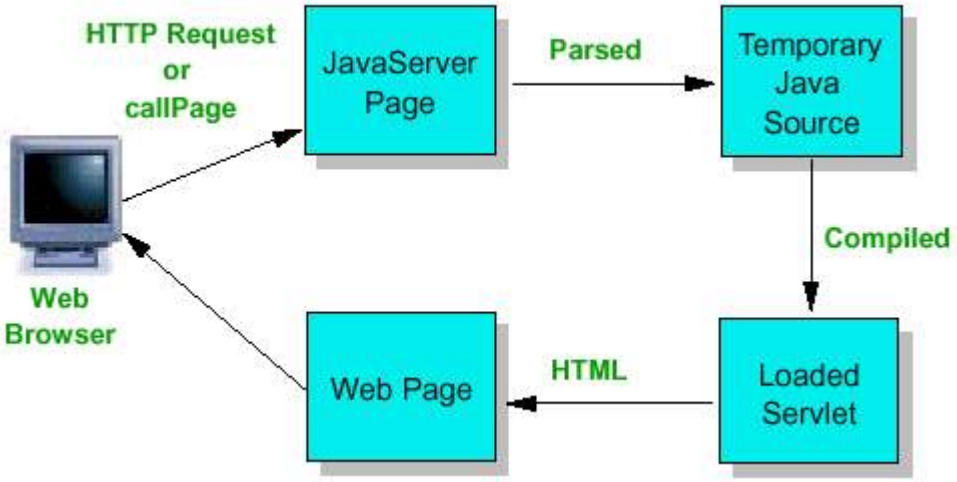


Figura 7.4.4: Como funciona una JavaServer Page.

7.5 Mantenimiento del estado de las aplicaciones Web.

HTTP es un protocolo sin estado, lo cual significa que no mantiene estados a través de los requerimientos de los clientes. En la mayoría de los casos, HTTP crea una nueva conexión para cada requerimiento, lo cual significa que no hay manera de que un servidor pueda reconocer que una serie de requerimientos han venido desde el mismo cliente. En muchas aplicaciones Web, la habilidad de mantener información a través de los requerimientos del cliente es uno de los principales puntos.

Existen algunas formas de añadir el estado al HTTP, incluír autentificación de un servidor Web, usar campos ocultos en una forma, las cookies, la reescritura de una URL. La administración de sesión de un servlet maneja el estado en un nivel más alto, y soporta las cookies y reescritura de la URL a través del API de Administración de Sesión.

7.5.1 Autentificación de un Servidor Web.

La mayoría de los servidores Web soportan autentificación de usuarios que restringen el acceso de los recursos a los usuarios que se han conectado usando un usuario y una clave mantenidos en el servidor. Además de limitar el acceso a los recursos, la autentificación puede ser usada para hacerle seguimiento a la sesión de un cliente. Cuando un usuario se conecta, el browser retiene el usuario y la clave y las reenvia con cada requerimiento al mismo realm. Un realm es la combinación de la localidad o recurso y el nombre del host del servidor.

Esta opción es simple de implementar y es soportada en la mayoría de los servidores Web. Además, mientras el browser no sea reiniciado, tratará de reusar el mismo usuario y la misma clave en cualquier momento que se vuelva a intentar accesar el mismo realm.

El mayor problema con esta opción es que cada persona necesita tener un único usuario y clave cuando se conecte cada vez que visite un sitio. Esto puede no gustarles a las personas, ya que esperan que sólo les pida usuario y clave para información restringida, pero no siempre que se conecten. Además, mientras todas las conexiones de un mismo usuario se consideran una sesión, estas pueden venir desde diferentes máquinas.

7.5.2 Campos ocultos en las formas.

Los campos ocultos en las formas, como su nombre lo indica, son campos en una forma HTML que no se muestran en el browser del cliente. Son enviados al servidor cuando la forma HTML que los contiene es enviada. Estos campos pueden ser usados para mantener información de estado poniendo un identificador de sesión oculto cada vez que la respuesta es enviada el cliente.

Este método es soportado en la mayoría de los browsers, no requiere seteo de los servidores ni que el usuario esté conectado. El mayor problema es que sólo funciona con formas

generadas dinámicamente. Si el sitio Web tiene formas estáticas y dinámicas, este método no es el apropiado.

7.5.3 Cookies.

Una cookie es una porción de dato pasado entre un servidor Web y un browser. El servidor envia una cookie que contiene datos los cuales son requeridos la próxima vez que el browser accese el servidor. Estos datos pueden identificar el usuario, al servidor Web, además de almacenar otra información. Cuando el browser recibe la cookie, la almacena, y la reenvia al servidor cuando la requiera.

El principal problema con este método es que no todos los browsers aceptan las cookies. Un browser rechaza las cookies debido a que el usuario no las desea. Muchos usuarios son suspicaces de todo lo que se almacene en sus máquinas de fuentes externas a ellos.

7.5.4 Reescritura de la URL.

Otra manera de seguir la información del usuario es anexar información del estado a los hiperenlaces en cada página enviada al browser. Esta técnica es conocida como reescritura de la URL. Cuando el browser hace un nuevo requerimiento al servidor, el requerimiento contiene información sobre el cliente.

Una desventaja de reescribir la URL es que el usuario debe seguir una ruta estricta a través del sitio Web, no se puede desviar de páginas que tienen codificadas las URLs y el desarrollador debe ser cuidadoso al reescribir todas las URLs que son enviadas de regreso al cliente.

7.5.5 Administración de la sesión servlet.

Todos los métodos anteriores muestran la necesidad de añadir estado al protocolo HTTP y cada método tiene sus ventajas y desventajas. La API del Java servlet contiene tipos diseñados para manejar la sesión en un nivel más alto permitiendo a los desarrolladores enfocarse en la construcción de las aplicaciones Web.

Con la administración de la sesión del servlet, cada usuario puede ser asociado con objetos HttpSession que son usados para almacenar o recuperar información sobre el usuario. Este objeto mantiene información sobre una sola sesión. Otros objetos Java pueden ser añadidos a la sesión usando el método putValue y recuperados usando getValue. Un objeto de sesión puede ser creado y recuperado usando el método getSession.

El manejo de la sesión puede ser implementado usando cookies o reescritura de la URL. Un identificador único para el objeto de sesión es puesto en la cookie o añadido a la URL, el cual es usado para recuperar el objeto de sesión.

7.5.6 Ciclo de vida de una sesión.

Una sesión es una conexión contínua desde el mismo browser a través de un periodo continuo de tiempo. Una sesión puede ser terminada automáticamente por el servidor después de un periodo de tiempo, o manualmente por el servlet llamando al método invalidate.

Parte III: Diseño e Implementación de Aplicación Bancaria.

Capítulo 8: Ambiente CORBA.

Para ambientes distribuidos utilizando la tecnología CORBA existen varios productos. Para este caso hemos escogido IBM WebSphere y la implementación CORBA de Websphere.

WebSphere es una implementación liviana de las especificaciones de la OMG. Visibroker está en el otro lado y provee la más completa implementación disponible en la industria.

8.1 WebSphere.

Es un conjunto de programas que incluye WebSphere Aplication Server, WebSphere Performance Pack, y otros productos que ayudan a los clientes a manejar sitios web. Este combina un ambiente de ejecución para Java servlets con conectores a los formatos más comunes de base de datos, ORB estándar, y middleware empresariales.

8.1.1 Ambiente de Ejecución Servlet.

El ambiente de ejecución servlet también conocido como ServletExpress, provee una máquina rápida para ejecutar Java servlets y JavaBeans del lado del servidor. Los Java servlets pueden ser usados para coordinar E/S entre JavaBeans servidores y los clientes HTML. Los servlets facilitan la esquematización y separación de la lógica de presentación, la lógica del negocio, y la lógica del acceso a los datos. Los JavaBeans servidores son usados para la lógica del negocio y el acceso a la base de datos, las transacciones y las aplicaciones existentes.

8.1.2 Java Server Pages.

JSP permite a Java o a los JavaBeans del lado del servidor, ser integrados dentro de páginas HTML. JSP es análogo a como Active Server Pages permiten la integración de los ActiveX del lado del servidor. JSP hace posible mantener al servidor Java separado de las voluminosas HTML usadas por el GUI del browser. Así es como lo hace:

- JSP define tags que permiten a las páginas HTML llamar a los JavaBeans del lado del servidor.

- JSP define una Java Servlet API que permite a un servlet llamar a páginas JSP HTML.

Por ejemplo, cuando un servlet recibe un requerimiento (un click de un botón, por ejemplo) desde el browser cliente, este llama a uno o más JavaBeans del lado del servidor. Una vez que la respuesta es formulada usando una variedad de base de datos, transacciones o JavaBeans de la lógica del negocio, entonces:

- Los servlets puede pasar la respuesta al browser Web invocando a una página JSP HTML.

  - ✓ Sin JSP, los servlets necesitan formatear la página HTML dinámicamente dentro del mismo.

  - ✓ El código HTML generado dentro de los servlets es dificil de mantener.

- Accesar a los JavaBeans directamente desde las páginas JSP HTML es fácil, llamando a métodos o propiedades en java.

  - ✓ JSP permite al desarrollador Java enfocarse a escribir las datos transaccionales, o la lógica del negocio en Java.

  - ✓ JSP permite a los desarrolladores GUI enfocarse a escribir o generar HTML.

8.1.3 Desarrollo CORBA y Soporte en Tiempo de Ejecución.

Además de la comunicación CORBA servidor-a-servidor, Visual Age for java puede ser usado para desarrollar applets que interactúen con objetos Java del lado del servidor, manejados por WebSphere y accesados a través del siguiente soporte de CORBA:

- CORBA Naming Service.
- Callbacks Servidor-a-Cliente.
- Generación de código IDL: las interfaces de los objetos pueden ser definidas en Java o IDL..
- Activación automática del objeto servidor, basado en requerimientos servlets desde cualquier browser.
- Fácil creación de objetos del lado del servidor usando fábricas de objetos distribuidos.
- Objeto servidor genérico extensible: organiza grupos de objetos servidores dentro de los servlets.
- Paso de objetos por valor.
- Soporte de excepciones Java: provee transparencia a las excepciones de los objetos distribuidos.

8.1.4 WebSphere Performance Pack.

El Performance Pack permite a las aplicaciones servidores Web tener escalabilidad, disponibilidad y balanceo de la carga. Este combina el balanceo de la carga, manejar el cache, los proxy y las funciones de filtro, el manejo de contenidos de archivos, y la replicación dentro de una sencilla infraestructura.

Algunas plataformas incluidas son Windows NT, AIX y Sun Solaris. Una cantidad similar de funciones está disponible en OS/390 por Domino GO Webserver 5.0.

8.2 Implementación CORBA de Websphere.

La implementación Java CORBA de Websphere es un completo ORB para CORBA 2.0 y soporta un ambiente de desarrollo para construir, desarrollar y manejar aplicaciones con objetos distribuidos que interactúan a través de varias plataformas. Los objetos construidos son fácilmente accesados por las aplicaciones Web que se comunican por medio del Internet Inter-ORB Protocol (IIOP) de la OMG. Este es el estándar de comunicación entre los objetos distribuidos ejecutándose en internet, en intranets y en ambientes de computación empresariales. Siendo una implementación nativa de IIOP, asegura alto rendimiento e interoperabilidad en las aplicaciones distribuidas.

8.2.1 Características de Websphere CORBA.

- *Repositorio de Interfaces (Interface Repository).* Es una base de datos en línea de la meta información de los tipos de objetos ORB. La meta información almacenada incluye información acerca de los módulos, las interfaces, las operaciones, los atributos y las excepciones.

- *Interfaz de Invocación Dinámica (Dynamic Invocation Interface).* Los programas clientes pueden obtener información acerca de una interfaz de un objeto del Repositorio de Interfaces, y dinámicamente construir el requerimiento que ejecutará el objeto.

- *Interfaz Esqueleto Dinámica (Dynamic Skeleton Interface).* Permite a los servidores atender las operaciones requeridas por el cliente que no fueron definidas estáticamente en la compilación.

- *Hilos de Ejecución y Administrador de Conexión*. Visibroker permite escoger entre dos políticas de hilos: hilos encolados (thread pooling). o paso.por.sección. Cuando se

selecciona la política de hilos, VisiBroker selecciona automáticamente la manera más fácil de manejar conexiones entre las aplicaciones de los clientes y los servidores.

- *Servicio de Localizacion (Location Service).* Es una extensión de las especificaciones CORBA que provee facilidad para localizar instancias de objetos. Trabaja con el Smart Agent sobre la red, y puede ver todas las instancias de un objeto enlazadas a los clientes.

- *Compilador idl2java*. El compilador idl2java convierte su código Java en IDL, generando los stubs clientes en el lenguaje que se escoja. Debido a que mapea las interfaces Java a IDL, se pueden reimplementar objetos Java en otro lenguaje de programación que soporte al mismo IDL. Incluye su pre-procesador basado en Java, el cual soporta argumentos estándares de pre-procesamiento.

- *Manejador de Eventos de Comunicación (Communication Event Handlers).* Visibroker permite a la aplicación cliente y a la implementación de objetos definir sus propios métodos para manejar las excepciones ORB y los procesos de recuperación. El mecanismo del manejador de eventos notifica a los clientes y a las implementaciones de objetos de eventos del sistema que pueden ser usados para implementar facilidades de seguimiento, depuración, login, seguridades y encriptación.

8.2.2 Websphere CORBA.

Las funciones que soportan la implementación ORB de Websphere son:

- *Runtime ORB*. Este runtime soporta la ejecución del programa cliente o servidor.

- *Componentes Servidores*. Estos componentes son instalados de manera opcional e incluyen el Repositorio de Interfaces.

Se puede usar el ORB en el desarrollo e implementación de programas que funcionan tanto en clientes como en servidores.

8.2.3 Desarrollando Aplicaciones con CORBA Websphere.

Al crear una aplicación con CORBA Websphere se pueden seguir los siguientes pasos:

- Especificar todos los objetos y sus interfaces usando el IDL. El IDL puede ser mapeado a una variedad de lenguajes de comunicación.

- La interfaz es usada por el compilador idl2java (com.ibm.IDL.toJava.Compile) de Websphere para generar el código stub para el programa cliente y el código skeleton para la implementación del objeto.

- Usar el código skeleton para crear el servidor que implemente el objeto.

- El código para el cliente y el objeto, una vez completado, es usado como entrada del compilador Java para crear una aplicación o applet Java y un objeto servidor.

8.2.4 Implementando Aplicaciones con CORBA Websphere.

CORBA Websphere es también usado en la fase de implementación. Esta fase ocurre cuando los programas clientes y servidores han sido probados y están listos para producción. En este punto el administrador está listo para implementar los programas clientes en los equipos del usuario final y las aplicaciones servidores en las máquinas servidores.

Se deben instalar las librerías (runtime) ORB en cada máquina el la que se ejecute el programa cliente. Debe existir el ORB completo en cada máquina en la cual se ejecutan las aplicaciones servidores.

Capítulo 9: Diseño de la Aplicación Bancaria.

En este capítulo se explicará cómo fue diseñada la Aplicación Bancaria en Casa.

9.1 Requerimientos.

Los requerimientos los hemos divido en dos: los requerimientos de las opciones que debe brindar la aplicación al cliente; y los requerimientos de la aplicación en cuanto a los productos o herramientas sobre la cual funciona.

9.1.1 Requerimientos del Cliente.

Los clientes que usen la Aplicación Bancaria deben estar en posibilidad de:

- Accesar a sus saldos de cuentas corrientes y de ahorros.
- Accesar las historias de los movimientos de sus cuentas.
- Transferir fondos entre sus cuentas.
- Crear cuentas.
- Transacciones de débitos y retiros de fondos de sus cuentas.
- Cambiar su clave de acceso o datos personales.

Los clientes deben poder hacer estas transacciones en forma segura y desde cualquier computadora conectada a Internet. También decidimos que los clientes deberían autentificarse por segunda vez cuando se haga una transacción que efectúe cambios en los saldos de las cuentas.

9.1.2 Requerimientos de la aplicación.

Los requerimientos iniciales, basados en los requerimientos de la aplicación son:

Un cliente de la aplicación necesita:

- Un código de usuario.
- Una clave de acceso.
- Al menos una cuenta activa en el banco.
- Un browser de Internet.
- Acceso a Internet.

Un proveedor de los servicios implementados por la Aplicación Bancaria necesita:

- Un servidor de aplicación.
- Un servidor Web.
- Un servidor de servlets.
- Soporte para CORBA.

9.2 Casos de Uso.

Un conjunto de casos de uso fue creado para la Aplicación Bancaria. El modelo completo de casos de uso se muestra en la Figura 9.2. Los casos individuales fueron usados como entradas al diseño de la arquitectura del modelo de objetos. Los casos individuales se encuentran en el Anexo A, “Casos de uso de la Aplicación Bancaria “. El modelo Casos de Uso es muy importante en una aplicación Web. Puede mapear muy de cerca a las páginas Web que conformarán el sitio Web y pueden servir como un resumen de los pasos que se sigan al navegar el sitio Web.

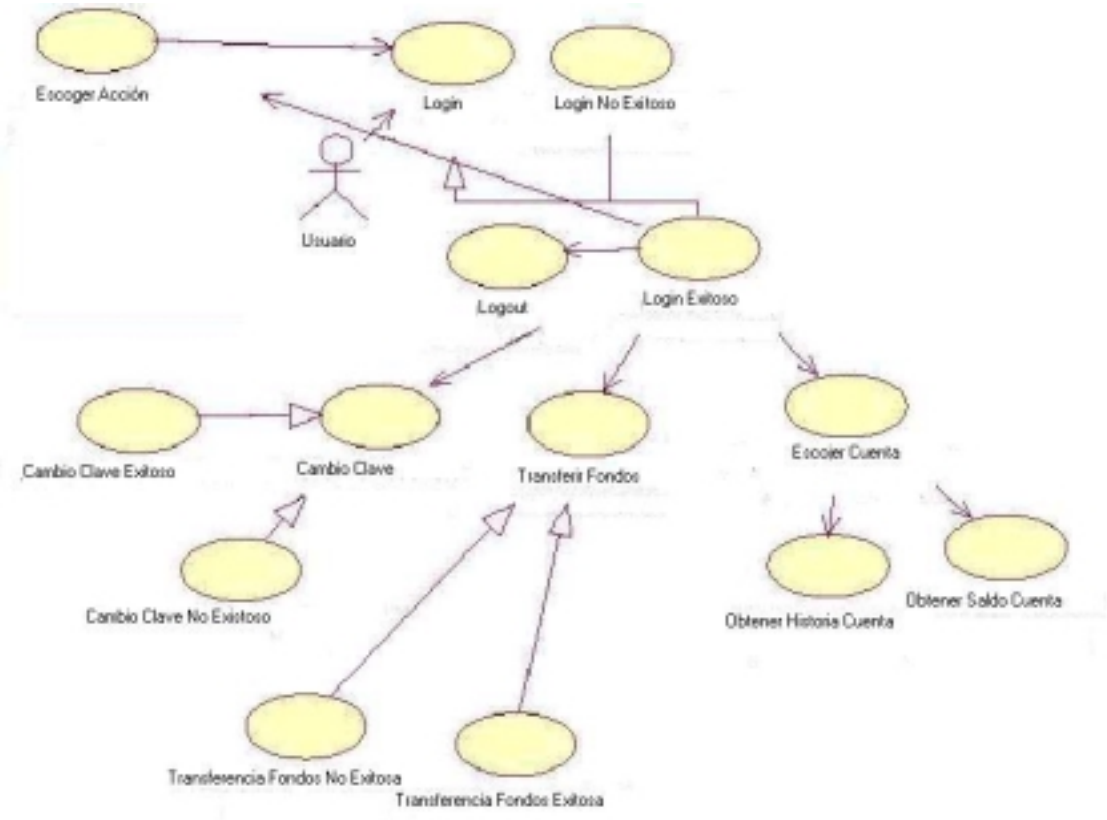


Figura 9.2: Modelo de Casos de Uso de la Aplicación Bancaria.

9.3 Diseño de la Base de Datos.

9.3.1 Relaciones.

En la implementación de la base de datos se han considerado las siguientes tablas y sus relaciones:

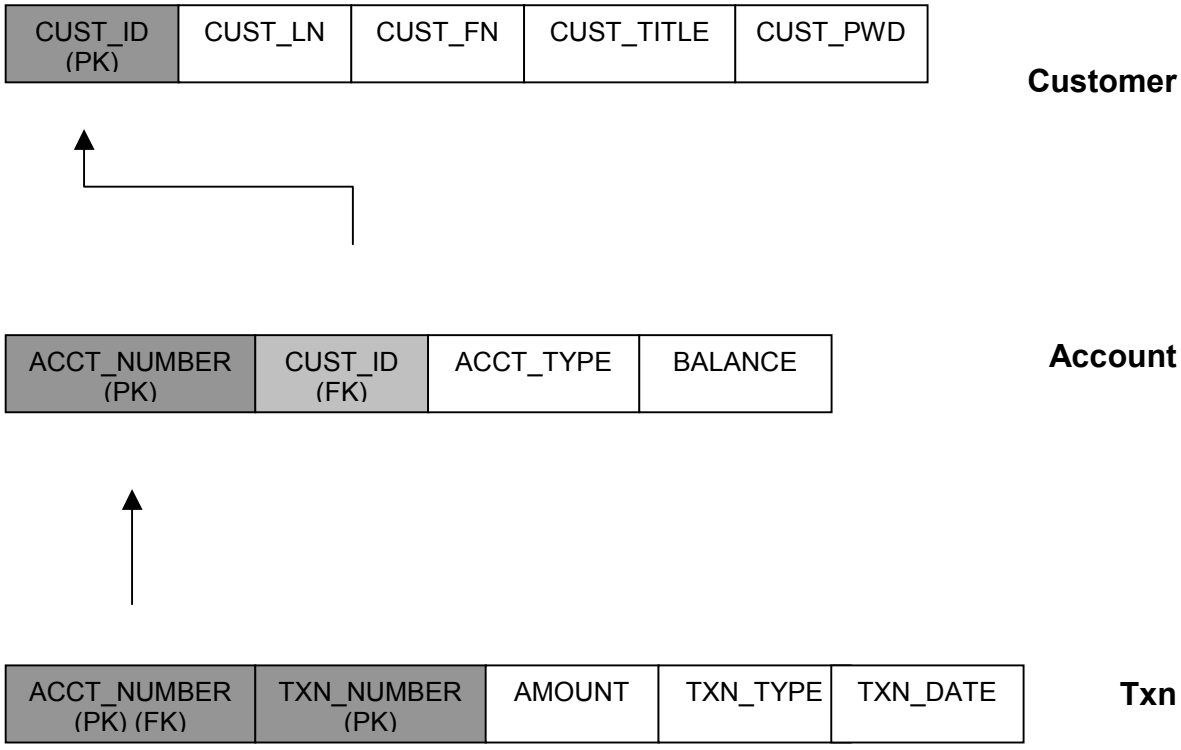


Figura 9.3: Relaciones entre tablas

9.3.2 Diseño físico.

A continuación se detalla el diseño físico de las tablas:

Tabla 1: Tabla CUSTOMER

COLUMNA	TIPO	TAMAÑO	CLAVE	NULOS	COMENTARIO
CUST_ID	CHAR	10	SI	NO	Identificación del cliente
CUST_LN	CHAR	30	NO	NO	Apellido del cliente
CUST_FN	CHAR	30	NO	NO	Nombre del cliente
CUST_TITLE	CHAR	5	NO	SI	Título del cliente
CUST_PWD	CHAR	10	NO	NO	Clave del cliente

Tabla 2: Tabla ACCOUNT

COLUMNA	TIPO	TAMAÑO	CLAVE	NULOS	COMENTARIO
ACT._NUMBER	CHAR	10	SI	NO	Número de cuenta
CUST_ID	CHAR	10	NO	NO	Identificación del cliente
ACCT_TYPE	CHAR	3	NO	NO	Tipo de cuenta
BALANCE	DEC	10,2	NO	NO	Saldo de la cuenta

Tabla 3: Tabla TXN

COLUMNA	TIPO	TAMAÑO	CLAVE	NULOS	COMENTARIO
ACCT_NUMBER	CHAR	10	SI	NO	Número de cuenta
TXN_NUMBER	INTEGER		SI	NO	Número de transacción
AMOUNT	DEC	10,2	NO	NO	Monto de transacción
TXN_TYPE	CHAR	3			Tipo de transaccion
TXN_DATE	DATE		NO	NO	Día de transacción

9.4 Diseño de la aplicación.

El primer paso en el desarrollo de las aplicaciones es crear el diseño basado en los requerimientos que la aplicación debe satisfacer. La importancia de un buen diseño determina la cantidad de esfuerzo que necesitaría para su implementación.

Capas de la aplicación.

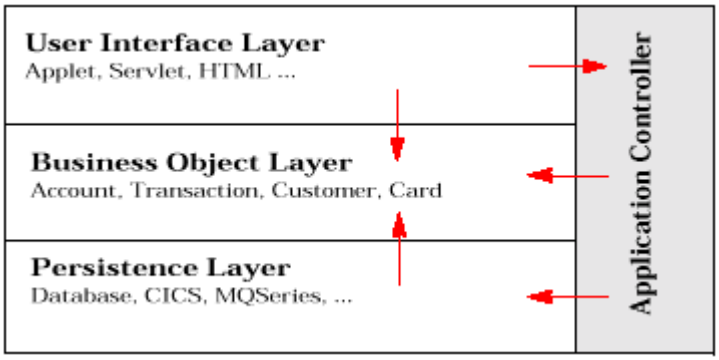
Un pre-requisito para aplicaciones a las que se pueda dar mantenimiento es una arquitectura de capas que asigne responsabilidades para ciertos servicios en una capa apropiada de aplicación. Estas responsabilidades son similares a un modelo de objetos, donde se asignan responsabilidades a cada objeto.

Primero toda la lógica del negocio y la base de conocimiento deberían ser modelados en objetos de negocios localizados en la capa del mismo nombre. Estos objetos representan entidades de la aplicación de banca y tienen la responsabilidad de su correcto funcionamiento. Para satisfacer las necesidades de las entidades arriba mencionadas, los objetos del negocio implementan métodos y propiedades en una vía estándar.

Luego se define una capa de interfaz de usuario (GUI) para separarla de los objetos del negocio. Los objetos GUI tienen la responsabilidad de manejar todas las interacciones del cliente y presentar los objetos del negocio en una forma que el cliente pueda observar.

También separamos el acceso a los datos del resto de la aplicación. La aplicación de banca debe manejar información de los clientes, las cuentas y las transacciones.

Figura 9.3: Controlador de la Aplicación.



El controlador de aplicación es necesario para conectar las capas. Básicamente hay tres maneras de conectar las capas:

- Cada objeto tiene el conocimiento requerido para accesar a otros objetos.

- Un ambiente de trabajo provee las interfaces y servicios necesarios. Los objetos de la aplicación se conectan al ambiente e interactúan entre sí.

- Las interfaces son modeladas en objetos con algo de conocimiento en ambos lados.

La aplicación, por su tecnología distribuida, se encaja en esta última manera de conectar las capas.

9.5 Objetos del negocio.

De los requerimientos antes mencionados podemos concluir que necesitamos dos tipos de cuentas, corriente y ahorro. Las principales funciones del negocio son identificación del cliente y las transacciones con sus cuentas. En lo que respecta a nuestra implementación requerimos las siguientes clases:

- Account. Permite manejar las cuentas.
- Customer. Permite la identificación del cliente.
- Transaction. Toda transacción debe ser grabada.

Los siguientes tipos de transacciones son definidas: depósito, retiro, transferencia. Adicionalmente la información del cliente, la verificación de password, y la historia de las transacciones también deben ser mantenidas.

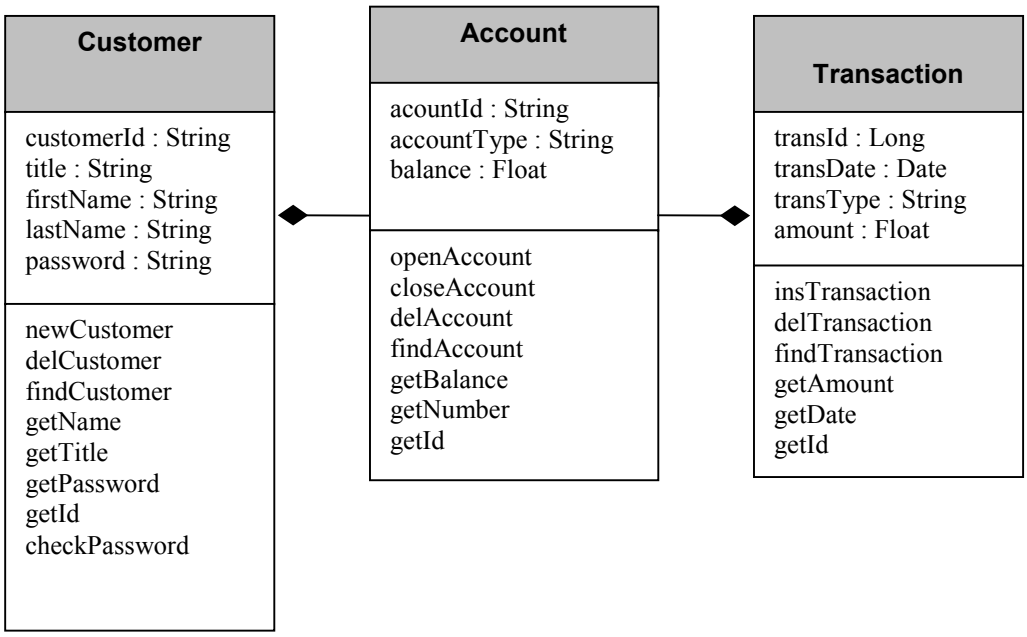


Figura 9.4: Modelo de Objetos.

9.6 Análisis de la aplicación.

En muchas aplicaciones Web el análisis del modelo debería basarse en el modelo de una aplicación preexistente. Especialmente en el caso de un sistema bancario que usa Internet, no debería construirse un sistema nuevo sin basarse en una infraestructura ya existente.

Debido a que no empezamos con una aplicación existente creamos un análisis del modelo de un modelo de negocios bancario. El modelo de objetos contiene tres elementos principales: Cuenta, Cliente, Transacción. Un Banco tiene muchas Cuentas que pueden ser de dos tipos: Corriente, Ahorros. Sobre Cuenta se pueden hacer varias transacciones por lo tanto debe tener varias opciones cada una de las cuales constituye una Transacción. Cada Cuenta tiene un propietario, un Cliente.

9.6.1 Modelo de Seguridad.

- El acceso al servidor Web será controlado a través de claves y un entorno de seguridad.

- Los usuarios requerirán conectarse a la Aplicación e ingresar una clave para inicializar cualquier transacción.

9.6.2 Arquitectura de la aplicación.

Ahora empezamos con el diseño de la aplicación. Conocemos la estructura de la página y el flujo del modelo de casos posibles, tenemos un modelo de objetos y las opciones de la aplicación.

La arquitectura básica de la Aplicación Bancaria se define por el alcance del proyecto: construir una Aplicación Bancaria usando herramientas de IBM que se ejecutan en el WebSphere Application Server.

9.7 Desarrollo CORBA del lado del servidor.

Websphere consiste de una implementación CORBA 2.0 y una colección de algunos servicios CORBA. El modelo de comunicación es levemente diferente a los demás proveedores de ORB en los que se usa applet-a-servlet. Cada servlet administra una o más intancias de objetos servidores en un contenedor. Un contenedor puede tener varios homes asociados; cada uno de los cuales puede crear, localizar y borrar instancias de objetos.

9.7.1 IIOP vs HTTP.

Algunas interacciones de Websphere usan IIOP para comunicarse con los ORB que están manejando la instancia del objeto. Otras interacciones de Websphere usan HTTP para comunicarse con un servlet.

Una interacción IIOP incluye IIOP puro e IIOP sobre HTTP (comúnmente referido como tunneling). Estrictamente hablando, tunneling usa HTTP debajo, es decir que el requerimiento es ruteado directamente a un objeto ORB y no a un servlet del Web server.

Una interacción HTTP es donde el cliente ORB hace una conexión al Web server e invoca un servlet. El servlet retorna el resultado a través del Web server.

9.7.2 Inicialización del Servidor CORBA.

En una aplicación CORBA servidor típica escrita en Java, se desarrolla una clase que contiene un método main(). Sin embargo una aplicación servidor Websphere no tiene un método main(), este opera en un ambiente servlet.

La primera vez que un servlet es invocado (usualmente usando los métodos Ihome), el Web server crea una instancia del servlet e invoca al método init(). Este método es donde se ejecuta la secuencia de inicialización, y luego se invoca el método service() del servlet. Las invocaciones subsecuentes del Web server al servlet invocarán al método service().

Websphere provee un servlet llamado GenericObjectServlet que contiene la suficiente implementación para actuar como servidor CORBA sin desarrollo adicional, lo que da unas características extras a cualquier aplicación. Las subclases de GenericObjectServlet tienen dos métodos claves:

El método getImpls() sobrescribe el método que se encuentra en GenericObjectServlet para retornar una lista del mapeo entre los nombres de las interfaces y los nombres de las clases de implementación. Este método es llamado una sola vez durante la secuencia de inicialización del servlet.

El método init() sobrescribe el método que se encuentra en GenericObjectServlet. Debemos notar que este llama al método init() de la superclase antes de a si mismo. Esto es muy

importante en la inicialización del ORB. Asignamos la variable orb (de la superclase) a una variable estática de tal manera que podamos obtener el ORB más tarde desde otras clases, luego llamamos al método initORB(), el cual crea algunos objetos iniciales de la aplicación.

9.7.3 Inicialización de los servlets.

Una de las desventajas de usar servlets como servidores CORBA es que el método init() no es invocado hasta que el cliente use un Ihome para crear una referencia a un objeto ORB.

Si el cliente de la aplicación tuviese un IOR con el cual pueda referenciar al objeto ORB directamente, no necesitaría llamar al servlet. Entonces necesitaría código adicional de inicialización que debería ser ejecutado si no usa un Ihome, es decir debería forzar el método init() del servlet. Todo esto se puede solucionar añadiendo el servlet a la configuración del WebSphere y cambiarle la opción de cargar al inicializar.

9.7.4 Ciclo de vida.

Una parte fundamental en el desarrollo de una aplicación es manejar el ciclo de vida de los objetos del negocio. Tal manejo es de un interés particular en un ambiente orientado a objetos distribuidos, porque:

- La creación de un objeto puede ser iniciada desde cualquier número de clientes.
- El objeto por sí mismo reside en cualquier número de servidores.
- El objeto puede ser movido a través de los servidores.
- El objeto puede ser “borrado” desde cualquier cliente.

Cada una de estas operaciones debe ser asociada con las respuestas que se necesitan direccionar si se va a proveer una aplicación distribuida robusta.

Debemos notar que en las aplicaciones grandes, de escala empresarial, el enfoque debería estar basado en un framework que incorpore la persistencia, el ciclo de vida y el manejo de las transacciones.

Capítulo 10: Implementación de la Aplicación Bancaria.

La implementación de un prototipo es usado para explorar la apariencia Aplicación Bancaria, el cual fue creado usando NetObjects Fusion y Visual Age. La apariencia del sitio puede ser muy productiva debido a que se pueden crear plantillas rápidamente y estilos de sitios para mostrar diferentes ideas.

10.1 Implementando el IDL

Mapeando de IDL a Java.

Basándonos en el análisis del modelo de objetos, se diseñó el archivo IDL para cada uno de ellos.

```
 // file acct.idl
 module IDLDevelopment
 {interface ORBAccount
    { attribute string fieldNumber;
attribute string fieldCustomerId;
.
.
float deposit(in float howMuch, in string acctnum);
float withdraw(in float howMuch, in string acctnum);
.
.
string getBalanceArray(in long i);
boolean saveTrans(in string acctnum, in string type, in float
amount);
     };
 };
```

```
// file cust.idl
module IDLDevelopment
{interface ORBCustomer
    {attribute string fieldCustomerId;
    attribute string fieldFirstName;
     .
     .
    boolean   checkPassword(in   string   customerId,   in   string
password);
    string getGrettings(in string customerId);
     .
     .
    boolean findElement(in string customerId);
    };
};
```

```
// file trans.idl
module IDLDevelopment
{interface ORBTrans
    {attribute string fieldAcctNumber;
     attribute long fieldNumber;
      .
      .
     void getTransInfo(in string customerId);
     boolean createTrans(in string acctnumber, in long number, in
string type, in string date, in float amount);
      .
     .
     string getAmountArray(in long i);
    };
};
```

Luego se compila cada uno de ellos con las clases que nos provee la implementación de CORBA de Websphere y obtenemos los fuentes en Java de los objetos necesarios para cumplir con las especificaciones CORBA.

```
prompt> java com.ibm.IDL.toJava.Compile –allTIE acct.idl
```

Código Generado.

Compilando cada uno de los archivos idl se generan varios archivos con extensión java. Estos archivos son almacenados en el subdirectorio generado IDLDevelopment, el cual es el nombre del módulo en el archivo IDL. Las interfaces son mapeadas como interfaces en el paquete. Por ejemplo para el archivo acct.idl tenemos los siguientes archivos:

- ORBAccount.java: declara la interfaz ORBAccount a través de la cual se accesará al objeto.

- ORBAccountHelper.java: declara la clase ORBAccountHelper la cual contiene funciones útiles de ayuda

- ORBAccountHolder.java: declara la clase AccountHolder la cual provee un retenedor para pasar parámetros.

- _ORBAccountStub.java: código Stub para el objeto Account en el lado del cliente.

- _ORBAccountSkeleton.java: código Skeleton para la implementación del objeto Account en el lado del servidor.

10.2 Implementando la lógica del negocio.

Se implementaron todos los objetos del negocio con sus respectivos métodos y propiedades como parte del paquete IDLDevelopment, el cual contiene las siguientes clases de la lógica del negocio:

- SimpleAccount.
- SimpleCustomer.
- SimpleTrans.

Clase SimpleCustomer.

Esta clase mantiene la información acerca del cliente, esto es: una identificación, título, nombre, apellido y una clave. Tiene las siguientes propiedades:

- customerId, type java.lang.String.
- title, type java.lang.String.
- firstName, type java.lang.String.
- lastName, type java.lang.String.
- password, type java.lang.String.

La implementación de estas propiedades dan como resultado funciones que cambian u obtienen el valor almacenado en las variables globales para cada campo. Por ejemplo, la propiedad customerId es representada por el atributo fieldCustomerId y los métodos para cambiar u obtener su valor, fieldCustomerId(String s) y fieldCustomerId().

Luego se implementan los siguientes métodos:

- Un metodo checkPassword para verificar la clave del usuario:
  boolean checkPassword(in string customerId, in string password)

- Un método getGreetings para retornar el valor formateado del nombre del cliente:
  string getGrettings(in string customerId)

- Un método createCustomer para crear un nuevo cliente:
  boolean createCustomer(in string customerId, in string password, in string firstName, in string lastName, in string title)

- Un método removeCustomer para eliminar a un cliente:
  boolean removeCustomer(in string customerId)

- Un método updateCustomer para actualizar los datos de un cliente:

  boolean updateCustomer(in string customerId, in string password, in string firstName, in string lastName, in string title)

- Un método findElement para buscar a un cliente:

  boolean findElement(in string customerId)

Clase SimpleTrans.

El objeto SimpleTrans es el contenedor de los datos de las transacciones. Tiene las siguientes propiedades:

- accountId, type java.lang.String.
- number, type java.lang.int.
- amount, type java.math.BigDecimal.
- type, type java.lang.String.
- date, type java.sql.Sate.

La propiedad *accountId* identifica la cuenta, *number* setea una clave única para la transacción en la base de datos. La propiedad *amount* es el monto de la transacción. La propiedad *type* indica el tipo de la transacción: débito o crédito.

La implementación de estas propiedades dan como resultado funciones que cambian u obtienen el valor almacenado en las variables globales para cada campo. Por ejemplo, la propiedad customerId es representada por el atributo fieldNumber y los métodos para cambiar u obtener su valor, fieldNumber(int n) y fieldNumber().

Luego se implementan los siguientes métodos:

- Un método getTransInfo para obtener información acerca de los movimientos de la cuenta y almacenarlos en un arreglo de transacciones:

  void getTransInfo(in string acctnumber)

- Un método createTrans para crear una nueva transacción:

  boolean createTrans(in string acctnumber, in long number, in string type, in string date, in float amount)

- Un método removeTrans para eliminar una transacción:

  boolean removeTrans(in string acctnumber, in long number)

- Un método findElement para buscar una transacción:

  boolean findElement(in string acctnumber, in long number)

- Un método setTransNumber para cambiar el valor de Number en un elemento del arreglo de transacciones:

  void setTransNumberArray(in string s, in long i)

- Un método getTransNumber para obtener el valor de Number de un elemento del arreglo de transacciones:

  string getTransNumberArray(in long i)

- Un método setTransTypeArray para cambiar el valor de Type en un elemento del arreglo de transacciones:

  void setTransTypeArray (in string s, in long i)

- Un método getTransTypeArray para obtener el valor de Type de un elemento del arreglo de transacciones:

  string getTransTypeArray (in long i)

- Un método setTransDateArray para cambiar el valor de Date en un elemento del arreglo de transacciones:

  void setTransDateArray (in string s, in long i)

- Un método getTransDateArray para obtener el valor de Date de un elemento del arreglo de transacciones:

  string getTransDateArray (in long i)

- Un método setAmountArray para cambiar el valor de Amount en un elemento del arreglo de transacciones:

  void setAmountArray (in string s, in long i)

- Un método getAmountArray para obtener el valor de Amount de un elemento del arreglo de transacciones:

  string getAmountArray (in long i)

Clase SimpleAccount.

El objeto SimpleAccount es el manejador de las cuentas. Tiene las siguientes propiedades:

- accountId, type java.lang.String
- customerId, type java.lang.String
- type, type java.lang.String
- balance, type java.math.BigDecimal

La propiedad accountId identifica la cuenta en el banco, la propiedad customerId identifica al cliente al que pertenece la cuenta, la propiedad type indica el tipo de cuenta y la propiedad balance guarda el saldo actual de la cuenta.

Adicionalmente tenemos los siguientes métodos:

- Un método getAccountInfo para obtener información acerca de las cuentas que un cliente mantiene en el Banco y la almacena en un arreglo de cuentas:

  void getAccountInfo (in string acctnumber)

- Un método deposit para realizar un depósito de fondos a una cuenta específica:

  float deposit(in float howMuch, in string acctnum)

- Un método withdraw para realizar un retiro de fondos de una cuenta específica:

  float withdraw(in float howMuch, in string acctnum)

- Un método transfer para realizar una transferencia de fondos entre dos cuentas específicas del cliente:

  float transfer(in float howMuch, in string acctnum1, in string acctnum2)

- Un método saveTrans para grabar una transacción al momento de hacer un depósito, un retiro o una transferencia de fondos:

  boolean saveTrans(in string acctnum, in string type, in float amount)

- Un método createAccount para crear una nueva cuenta:

  boolean createAccount(in string acctnum, in string customerId, in string type, in float balance)

- Un método removeAccount para eliminar una cuenta:

  boolean removeAccount(in string acctnum)

- Un método findElement para buscar una cuenta:

  boolean findElement (in string acctnum)

- Un método setAccountNumber para cambiar el valor de Number en un elemento del arreglo de cuentas:

void setAccountNumberArray(in string s, in long i)

- Un método getAccountNumber para obtener el valor de Number de un elemento del arreglo de cuentas:
  string getAccountNumberArray(in long i)

- Un método setAccountTypeArray para cambiar el valor de Type en un elemento del arreglo de transacciones:
  void setAccountTypeArray (in string s, in long i)

- Un método getAccountTypeArray para obtener el valor de Type de un elemento del arreglo de cuentas:
  string getAccountTypeArray (in long i)

- Un método setAccountDateArray para cambiar el valor de Date en un elemento del arreglo de cuentas:
  void setAccountDateArray (in string s, in long I)

- Un método setBalanceArray para cambiar el valor de Balance en un elemento del arreglo de cuentas:
  void setBalanceArray (in string s, in long i)

- Un método getBalanceArray para obtener el valor de Balance de un elemento del arreglo de cuentas:
  string getBalanceArray (in long i)

10.3 Implementación de los servlets.

Para implementar los requerimientos de objetos distribuidos CORBA, se utiliza un servlet por cada objeto, el cual tiene la finalidad de levantar la parte servidor CORBA y permitir el acceso

a los respectivos objetos, estos son EspolAccountServlet, EspolCustomerServlet, EspolTransServlet.

Para el manejo de la interfaz del usuario se crea un servlet administrador llamado BankAdminServlet.

Account Servlet.

Después de validar la clave del usuario, éste puede escoger visualizar la información de las cuentas o manipular sus fondos. El servlet para cuentas (clase AccountServlet) permite, al ser invocada, levantar los servicios de CORBA tales como el skeleton, stub, helper y la misma implementación para que a través de una interfaz ORBAccount el cliente pueda acceder a las propiedades y métodos del objeto implementado.

Transaction Servlet.

El servlet para transacciones (clase TransServlet) permite, al ser invocada, levantar los servicios de CORBA tales como el skeleton, stub, helper y la implementación del objeto SimpleTrans para que a través de la interfaz ORBTrans el cliente pueda acceder a las propiedades y métodos del objeto implementado.

Customer Servlet.

El servlet para clientes (clase CustomerServlet) permite, al ser invocada, levantar los servicios de CORBA tales como el skeleton, stup, helper y la implementación del objeto SimpleCustomer para que a través de la interfaz ORBCustomer el cliente pueda acceder a las propiedades y métodos del objeto implementado.

BankAdmin Servlet

El servlet para la administración de la interfaz gráfica (clase BankAdminServlet) es invocada para acceder a las diferentes opciones que maneja la aplicación. Entre estas opciones tenemos

Login: Esta opción es utilizada después de validar la clave del usuario para acceder a la página principal de la aplicación. En este momento también se guarda como una variable de sesión la identificación del usuario, que más tarde va a ser utilizada en otras opciones. Esta característica le da a la aplicación un ambiente multiusuarios, inclusive desde un mismo equipo.

Logout: Es utilizada cuando el usuario selecciona salir del sistema y elimina las variables de sesión creadas al momento de ingresar.

Account: Crea dinámicamente una página HTML con el applet respectivo, el cual mostrará la información de las cuentas del cliente, que es pasado como parámetro al mismo.

Transfer: Crea dinámicamente una página HTML con el applet respectivo, el cual permitirá realizar las transferencias de fondos entre dos cuentas del cliente, que es pasado como parámetro al mismo.

Personal: Crea dinámicamente una página HTML con el applet respectivo, el cual permitirá realizar actualizaciones a los datos personales del cliente, entre ellos la clave de ingreso. Así mismo la identificación del cliente es pasada como parámetro del applet.

Trans: Crea dinámicamente una página HTML con el applet respectivo, el cual mostrará la información de las transacciones o movimientos de la cuenta seleccionada, en la pantalla de cuentas por cliente. El número de la cuenta es pasado como parámetro del applet.

newaccount: Crea dinámicamente una página HTML con el applet respectivo, el cual permitirá crear una cuenta para el cliente, que es pasado como parámetro al mismo.

10.4 Implementando la interfaz con Applets.

Para implementar los requerimientos de las interfaces del usuario se crearon varios applets los cuales se encargan:

- Iniciar la conexión con los objetos CORBA.
- Pedir la información al usuario.
- Procesar la información utilizando los métodos y propiedades de los objetos CORBA.
- Mostrar los resultados al usuario.

Para este propósito se han diseñado los siguientes applets:

- Login.
- NewUser.
- AccountInfo.
- TransInfo.
- Transfer.
- Request.
- Personal.

Login.

Pide el código del usuario (login) y la clave de acceso (password) correspondientes. Luego inicializa los servicios de CORBA para la interfaz del objeto Customer. A través de sus métodos valida la información y si es correcta llama al servlet administrador con la opción login para tener el acceso a la página principal de la aplicación.

NewUser.

Pide la información necesaria para crear un nuevo usuario de la aplicación. Esta información es el código del usuario, su password, su nombr , sus apellidos, el titulo del nuevo usuario, e

inicializa los servicios de CORBA para la interfaz del objeto Customer. A través de sus métodos crea el nuevo usuario y si todo es correcto llama al servlet administrador con la opción login para tener el acceso a la página principal de la aplicación.

AccountInfo.

Recibe como parámetro el código del cliente y muestra la información de sus cuentas. Inicializa los servicios de CORBA para la interfaz del objeto Account. A través de sus métodos obtiene información como el número, el tipo y el saldo de las cuentas activas en el banco. Adicionalmente se le da la opción al usuario de mostrar el detalle de sus cuentas, esto lo realiza llamando al servlet administrador con la opción trans para tener el acceso a los movimientos de las mismas.

TransInfo.

Recibe como parámetro el código de la cuenta y muestra la información de movimientos de la misma. Inicializa los servicios de CORBA para la interfaz del objeto Trans. A través de sus métodos obtiene información como el número, el tipo, la fecha y el monto de los movimientos realizados en esa cuenta.

Transfer.

Recibe como parámetro el código del cliente y permite transferir fondos entre las cuentas de un cliente. Inicializa los servicios de CORBA para la interfaz del objeto Account. Se selecciona la cuenta origen y la cuenta destino de la transferencia de los fondos, así como el monto a transferir. Si todo se realiza correctamente le mostrará el saldo resultante de la cuenta hacia la cual hizo la transferencia.

Request.

Recibe como parámetro el código del cliente y permite retirar o depositar fondos en las cuentas. Inicializa los servicios de CORBA para la interfaz del objeto Account. Se selecciona la cuenta a la cual se va a afectar y el monto de la transacción. Se escoge entre la opción de depósito o retiro y si todo se realiza correctamente le mostrará el saldo resultante de la cuenta.

Personal.

Recibe como parámetro el código del cliente y permite actualizar la información personal del cliente. La información que se puede actualizar es el password, el nombre, los apellidos, y el titulo. Inicializa los servicios de CORBA para la interfaz del objeto Customer. A través de sus métodos modifica la información del usuario y muestra un mensaje si todo es correcto.

10.4 Flujo de la aplicación.

Websphere, como servidor de aplicaciones, realiza las funciones de controlador de la aplicación, ya que permite dentro de su ambiente ejecutar los servlets mencionados anteriormente, los cuales controlan a su vez el flujo de la información entre las interfaces de loa objetos habilitadas por CORBA, el manejo de las interfaces o pantallas y la optimización en el acceso a la base de datos.

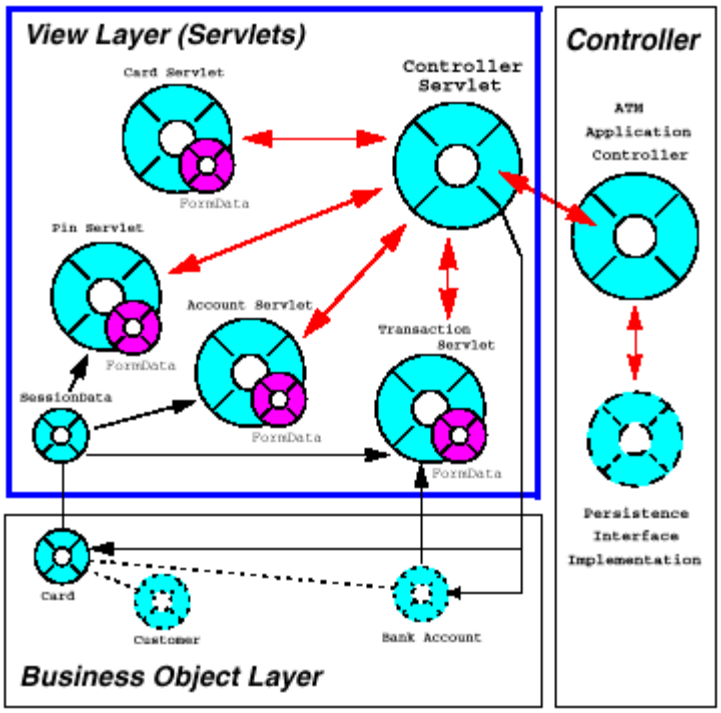


Figura 9.5: Capas de la Aplicación.

10.5 Implementando el Web Site.

Diseño del Web Site.

El Web site fue diseñado para facilitar el acceso y la navegación del usuario a través de la aplicación. De esta forma se diseñaron las siguientes páginas HTML:

- *Default.html* – Página inicial de la aplicación en la que se da la opción de firmar con usuario existente (login) o crear uno nuevo (sign-up).
- *Login.html* – Página que permite el ingreso al sistema.
- *NewUser.html* – Página para crear un nuevo usuario al sistema.
- *Bank.html* – Página principal del sistema, la cual ha sido confeccionada en tres partes (frames): la cabecera (logo.html), las opciones (index.html) y la página principal (main.html).
- *Index.html* – Página que contiene las opciones del sistema.

- *Logo.html* – Página que contiene la cabecera.

- *Main.html* – Página que contiene el mensaje introductorio
- *LoadJava.html* – Página que permite que la aplicación sea cargada.

10.6 Prototipo de la Aplicación Bancaria.

Un usuario empieza a interactuar con la aplicación bancaria ingresando a http://<nombreservidor>/hbespol/ en el browser y se le presenta la página inicial de la Aplicación Bancaria (Figura 9.6).

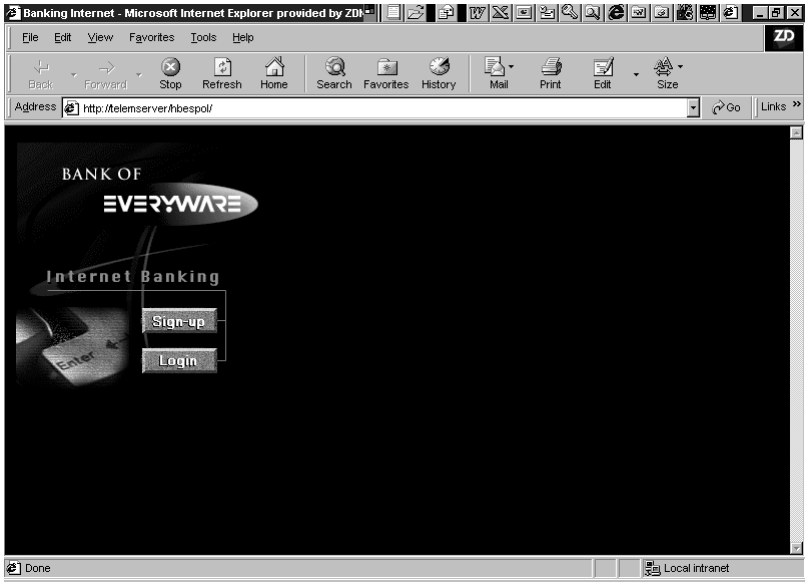


Figura 9.6: Página inicial de la Aplicación Bancaria.

Para accesar a sus cuentas, el usuario debe primero ir a la página de login para identificarse usando su Usuario y su Clave (Figura 9.7).

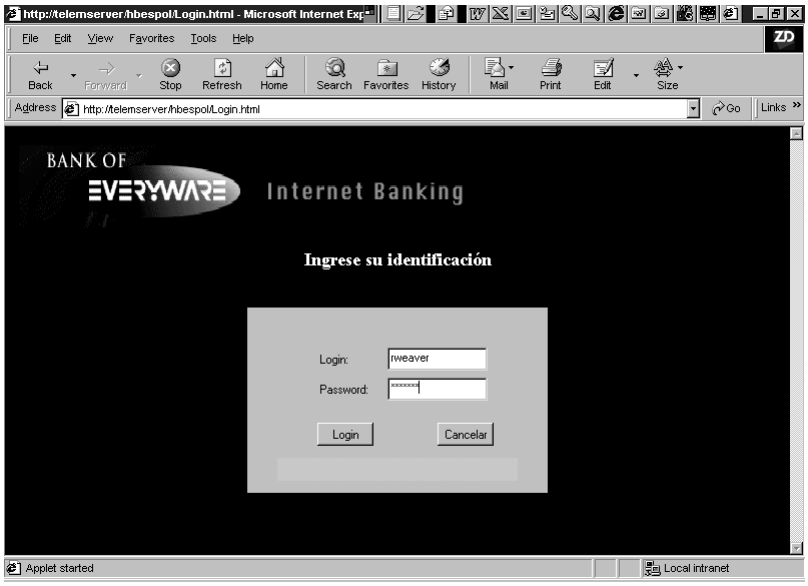


Figura 9.7: Página de login de la Aplicación Bancaria.

En el caso de no tener un usuario para accesar a la aplicación, entonces de ir a la página de signup para crear un usuario y clave (Figura 9.8).

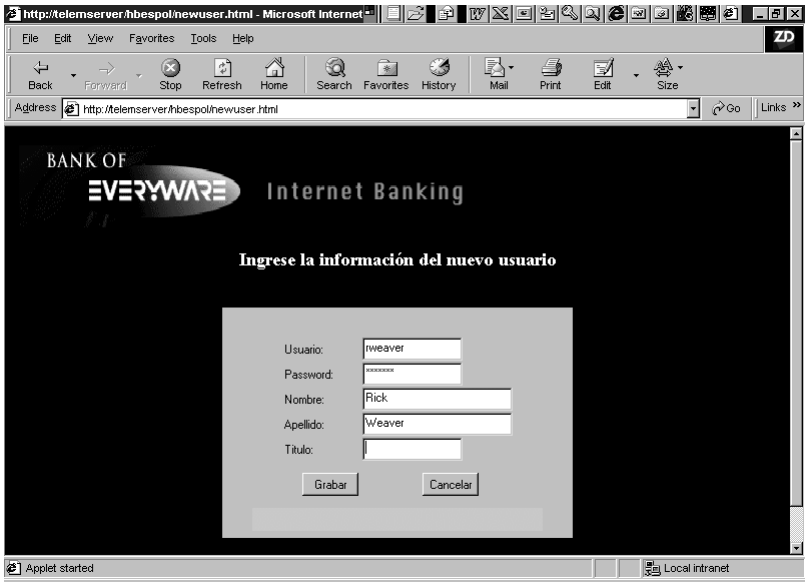


Figura 9.8: Página de sign-up de la Aplicación Bancaria.

Luego que se ha conectado, el cliente tiene total acceso a sus cuentas (Figura 9.9).

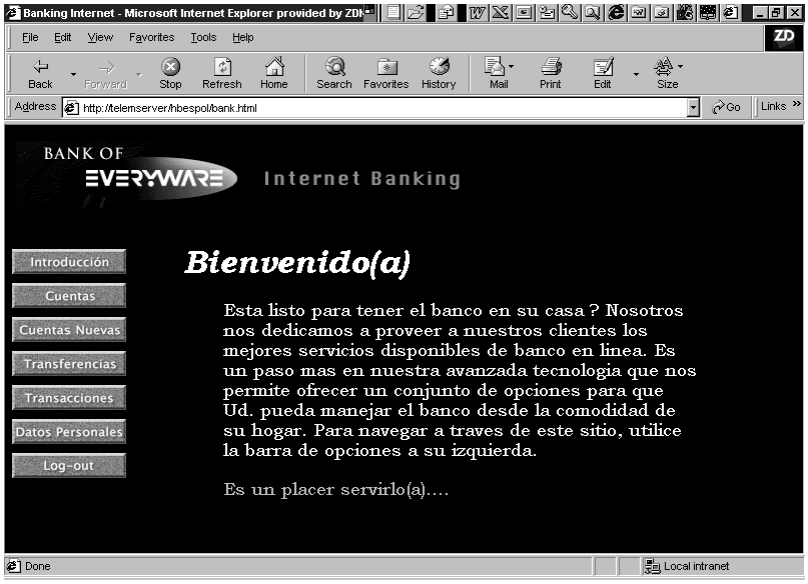


Figura 9.9: Pantalla principal de Aplicación Bancaria

A continuación describimos cada una de las opciones.

Logout.

El usuario puede escoger salir de la aplicación desde cualquier página en el sitio dando click desde el menú. Cuando se escoge salir el usuario envía el requerimiento al servlet administrador y la sesión es terminada.

Cuentas.

Esta opción provee los saldos y tipos de cuentas del cliente. Adicionalmente tiene la opción de ver los movimientos de cualquiera. Cuando el usuario requiere información de sus cuentas, el requerimiento es enviado al servlet administrador. Se levanta un applet que lista las cuentas del cliente en el banco junto con la opción de mostrar los movimientos de las cuentas (Figura 9.10).

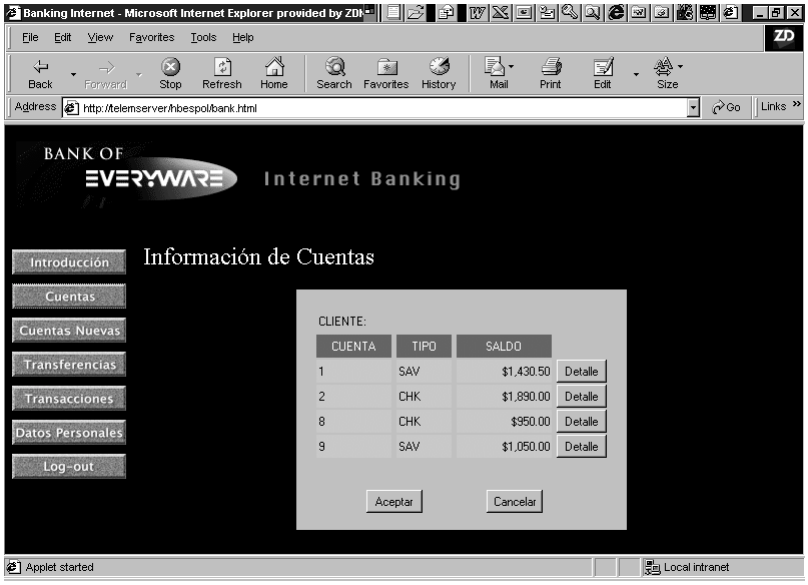


Figura 9.10: Información de Cuentas.

Movimientos de la Cuenta.

Cuando el usuario selecciona sus cuentas y da click en Detalle, el requerimiento es enviado al servlet administrador, el cual levanta un applet que lista las transacciones para la cuenta seleccionada (Figura 9.11).

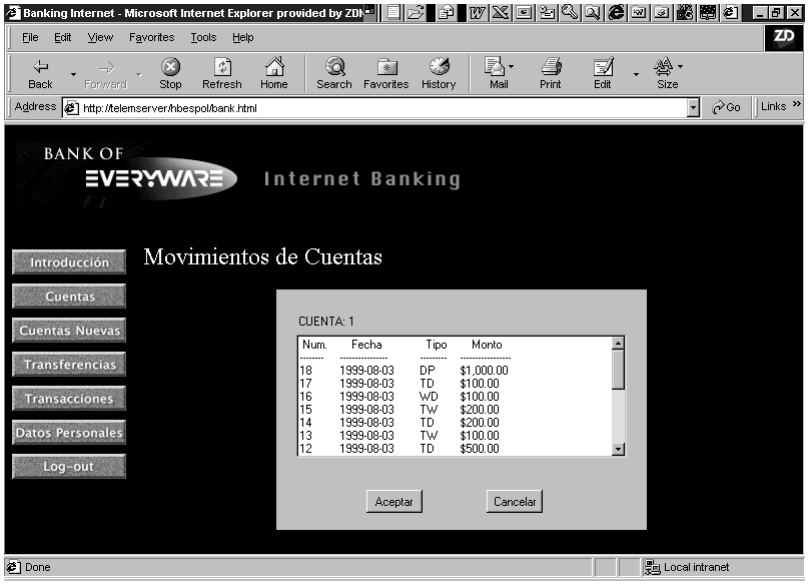


Figura 9.11: Movimientos de la Cuenta.

Cuentas Nuevas.

Cuando el usuario selecciona una cuenta nueva el requerimiento es enviado al servlet administrador el cual levanta un applet que permite crear una cuenta nueva para el cliente.

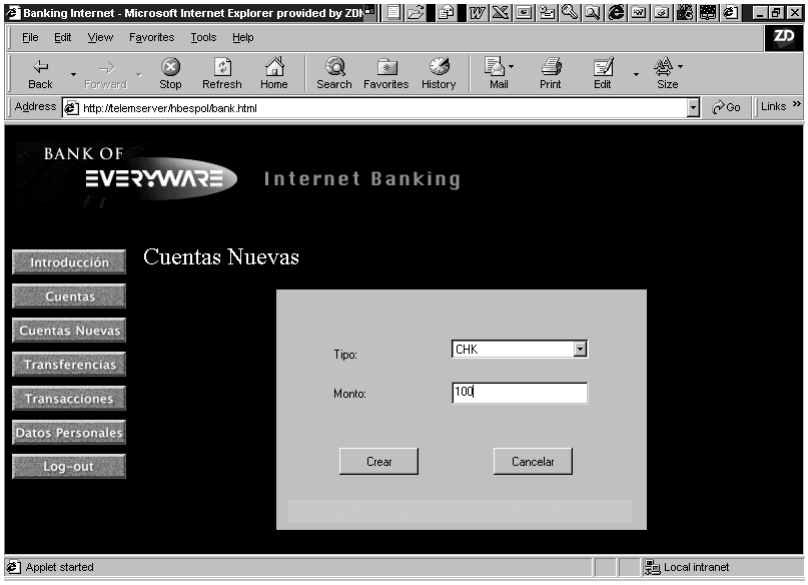


Figura 9.12: Cuenta Nueva.

Transferencia de Fondos.

Cuando el usuario selecciona esta opción el requerimiento es enviado al servlet administrador el cual levanta un applet que permite la transferencia de fondos. El usuario selecciona la cuenta origen y la cuenta destino de los fondos e ingresa el monto de la transacción y luego envía el requerimiento (Figura 9.13).

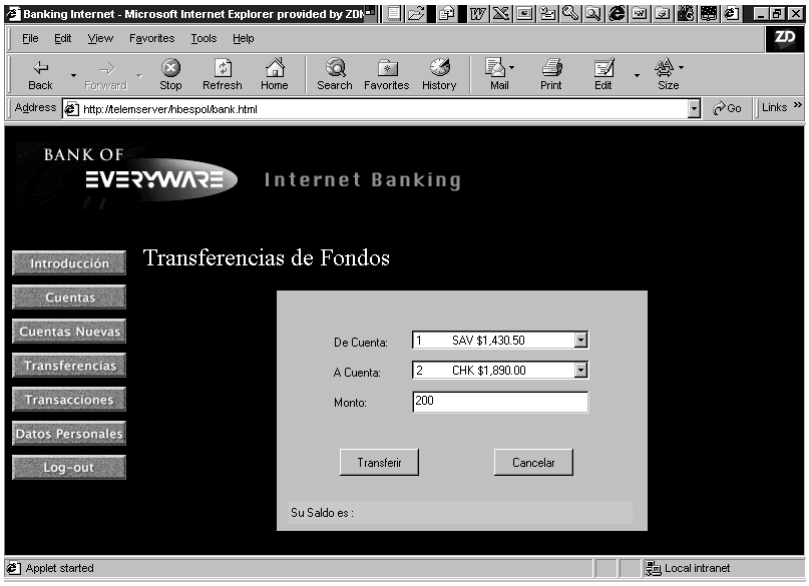


Figura 9.13: Transferencia de Fondos.

Transacciones Disponibles.

Cuando el usuario selecciona esta opción el requerimiento es enviado al servlet administrador el cual levanta un applet que permite realizar depósitos o retiros de fondos. El usuario selecciona la cuenta de la cual se va retirar o depositar los fondos e ingresa el monto de la transacción y luego envía el requerimiento (Figura 11.6.9).

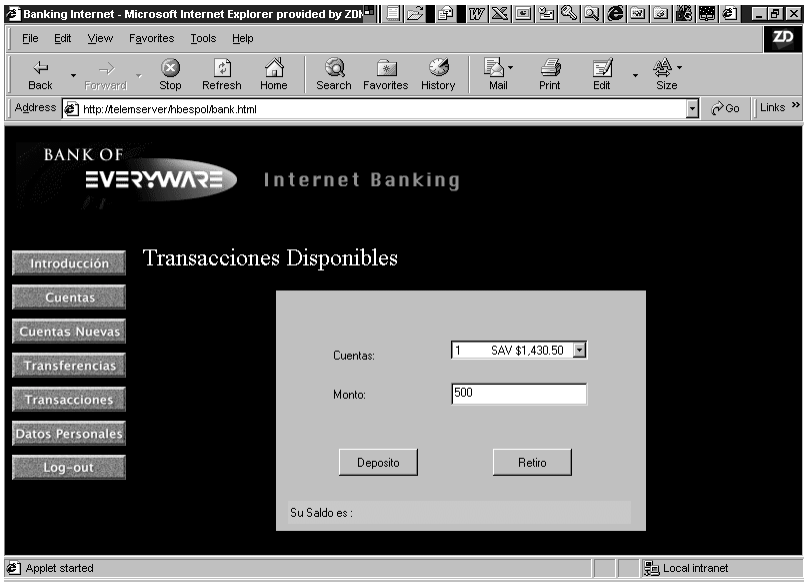


Figura 9.14: Transacciones Disponibles.

Datos Personales.

Cuando el usuario selecciona Datos Personales, el requerimiento es enviado al servlet administrador, el cual levanta un applet que permite actualizar la información del usuario de la aplicación, incluyendo el password de ingreso (Figura 9.15).

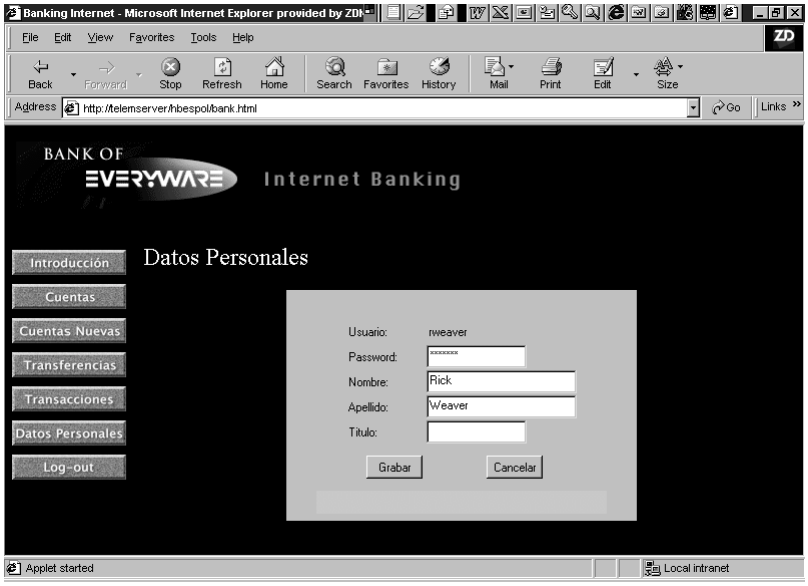


Figura 9.15: Datos Personales.

Capítulo 11: Distribución de la Aplicación Bancaria.

En este capitulo se explican todos los pasos hasta llegar a poner en producción la Aplicación Bancaria en ambos lados: cliente y servidor.

Las diferentes opciones del sistema fueron probadas usando Visual Age for Java. La distribución de la Aplicación Bancaria necesita un Web server, una máquina servlet y una implementación de CORBA. Se han escogido las siguientes plataformas: Windows NT para el servidor y Windows 95 para los clientes con Microsoft Internet Explorer 4.0 o superior y Netscape 4.5 o superior.

11.1 Configurando los Servidores.

Instalación de los Servidores.

- Se instaló como servidores: Web Lotus Domino Go Web Server e Internet Information Server siguiendo las instrucciones provistas por los respectivos manuales.
- Como servidor de aplicaciones se instaló WebSphere Application Server siguiendo los pasos que la documentación recomienda.

Configuración de los Servidores.

Para la implementación de CORBA se seleccionó la que provee WebSphere que implementa las especificaciones de la OMG de CORBA 2.0. Para su configuración se siguen los siguientes pasos:

1. Se deben copiar los archivos ibmjbde.jar e ibmjbrt.jar al directorio de clases del WebSphere Application Server.
2. Luego añadimos los archivos ibmjbde.jar e ibmjbrt.jar al classpath de WebSphere Application Server.

3. Configuramos el archivo ObjectImpl.properties ubicado en el directorio de clases de WebSphere Application Server y añadimos las implementaciones CORBA desarrolladas.

```
IDLDevelopment.ORBAccount=IDLDevelopment.SimpleAccount
IDLDevelopment.ORBCustomer=IDLDevelopment.SimpleCustomer
IDLDevelopment.ORBTrans=IDLDevelopment.SimpleTrans
```

11.2 Configuración de los Clientes.

Instalación de los Clientes.

- Se instaló como browser el Microsoft Internet Explorer 4.0 y Netscape 4.5 siguiendo las instrucciones provistas en sus respectivos manuales.

Configuración de los Clientes.

Para configurar CORBA del lado del cliente se siguen los siguientes pasos:

1. Se copian los archivos ibmjbde.jar e ibmjbrt.jar al directorio de clases del equipo. Este directorio puede ser alguno existente o uno nuevo.

2. Se añaden los archivos ibmjbde.jar e ibmjbrt.jar al classpath del sistema. Para esto se modifica el archivo autoexec.bat.

11.2 Distribución de la Aplicación.

En la aplicación bancaria tenemos un solo paquete con las clases necesarias para su funcionamiento. Los servlets también son clases individuales.

Distribución de los Applets.

El proceso básico de distribución de los applets es:

- Se copian los archivos .class al directorio asignado para la aplicación (/hbespol) en el directorio raíz del Web Server. Los archivos deben estar dentro del directorio IDLDevelopment ya que son parte del paquete de distribución.

- Se crea un archivo HTML para las clases applet Login y NewUser. Las páginas de las demás clases applets son generadas dinámicacmente por el servlet administrador.

Distribución de los Servlets.

Actualmente más Web servers contienen facilidades (run-time) que permiten la ejecución de los servlets. Algunos inclusive tienen su propia JVM y otros utilizan la del sistema. Los Web Server modernos han optimizado las facilidades como el cache que permiten mantener los servlets activos después de su primer uso. Esto aumenta el desempeño considerablemente, comparados con los programas CGI. El WebSphere contiene todas estas facilidades al momento de su instalación.

Debido a que los servlets se ejecutan en el Web server en un ambiente de confianza, no tienen restricciones acerca del acceso a otras máquinas.
El proceso básico de distribución de los servlets es:

- Obtener las clases servlets.
- Se copian los archivos al directorio de servlets (D:\WebSphere\AppServer\Servlets).

- Si es necesario, se deben setear las propiedades de los servlets del WebSphere.

Distribución del Sitio Web.

El Web site de la aplicación es manejado por NetObjects Fusion, el cual provee una herramienta de publicación automática del sitio. De otra forma también pueden ser publicadas manualmente, creando un subdirectorio en el directorio raíz del web server (\hbespol) y colocando los archivos html en el mismo. De igual forma se crea un subdirectorio IDLDevelopment correspondiente al paquete en el cual se colocarán las clases de la aplicación.

Si el Web server lo permite, se puede cambiar la configuración de la publicación del directorio para que la página inicial (default) sea default.html.

Localización de los archivos.

Para permitir el correcto funcionamiento de la aplicación debemos configurar el classpath y tener los archivos necesarios en sus directorios respectivos.

Para añadir los drivers DB2 JDBC necesarios para accesar a la base de datos:

```
D:\SQLLIB\java\db2java.zip;
```

Otras librerías necesarias son:

```
D:\jdk1.1.6\lib\classes.zip; <=JDK
D:\WebSphere\AppServer\web\classes; <= Directorio de Clases
```

CONCLUSIONES Y RECOMENDACIONES.

1.- En la época actual toda empresa debería contar con la mejor infraestrutura de IT (Information Technology) posible para poder enfrentar la competencia de las empresas rivales.

2.- La Aplicación de Home Banking es beneficiosa tanto para los Bancos como para los clientes, ya que permite que los Bancos puedan atraer clientes desde lugares lejanos, así como que los clientes puedan realizar transacciones bancarias desde la comodidad de sus respectivos hogares.

3.- Una de las metodologías más usadas actualmente por las empresas es la de e-business, la cual emplea la tecnología de Internet con sus elementos de HTML, XML, CORBA (Common Object Request Broker Architecture), etc., para poder cumplir un mejor desempeño y eficiencia al brindar servicios a sus clientes.

4.- La Aplicación Bancaria en Casa usa la arquitectura MVC (Modelo / Vista / Controlador), cuyo propósito es separar la lógica de la presentación de la lógica del negocio, para lo cual se usa un Controlador, el cual actúa como puente a través del cual se coordina la actividad entre la vista y el modelo.

ANEXOS.

Anexo A: Casos de uso de la Aplicación Bancaria.

Cada caso de uso indica una secuencia de acciones realizadas por el sistema en respuesta a un requerimiento de uno de los participantes en el sistema. Donde no se indique, el participante es el usuario.

CU01 – Login / Autentificación.

Participante: Usuario.

Definición: El usuario ingresa su código de usuario y su respectiva clave, y sumite lo ingresado. El sistema responde.

Usos / extensiones: ninguno.

CU01A – Login / Autentificación exitosos.

Participante: Usuario.

Definición: El usuario ingresa su código y su clave respectiva, sumite lo ingresado. El sistema responde con la pantalla del menú de cuentas.

Extensiones: Login / Autentificación.

Usos: ninguno.

CU01B – Login / Autenficación fallido.

Participante: Usuario.

Definición: El usuario ingresa su código y su clave respectiva, sumite lo ingresado. El sistema responde con la pantalla del error en el Login.

Extensiones: Login / Autentificación.

Usos: ninguno.

CU02 – Obtención del menú de cuentas.

Definición: El usuario escoge la opción de cuentas. El sistema responde con la pantalla de cuentas.

Extensiones: Ninguna.

Usos: Login / Autentificación.

CU03 – Obtención del saldo de una cuenta.

Definición: El usuario escoge la opción de Información de Cuenta. El sistema responde con la pantalla de Información de Cuenta. El usuario selecciona la opción de Saldo de la Cuenta. El sistema responde con la pantalla de Saldo de Cuenta.

Extensiones: Ninguna.

Usos: Login / Autentificación; menú de obtención de Cuentas.

CU04 – Obtención de la Historia de Cuenta.

Definición: El usuario escoge la opción de Información de Cuenta. El sistema responde con la pantalla de Información de Cuenta. El usuario selecciona una cuenta. El usuario selecciona la opción de Historia de Cuenta. El sistema responde con la pantalla de Historia de Cuenta.

Extensiones: Ninguna.

Usos: Login / Autentificación; menú de obtención de Cuentas.

CU05 – Transferencia de fondos entre cuentas del usuario.

Definición: El usuario escoge la opción de Transferencia de Fondos. El sistema responde con la pantalla de Transferencia de Fondos. El usuario selecciona la cuenta origen. El usuario selecciona la cuenta destino de los fondos. El usuario ingresa el monto de la transferencia. El Usuario ingresa la clave de la transacción. El usuario sumite los datos. El sistema responde.

Extensiones: Ninguna.

Usos: Login / Autentificación; menú de obtención de Cuentas.

CU05A – Transferencia exitosa de fondos entre cuentas del usuario.

Definición: El usuario escoge la opción de Transferencia de Fondos. El sistema responde con la pantalla de Transferencia de Fondos. El usuario selecciona la cuenta origen. El usuario selecciona la cuenta destino de los fondos. El usuario ingresa el monto de la transferencia. El Usuario ingresa la clave de la transacción. El usuario sumite los datos. El sistema responde con la pantalla de Fondos Transferidos mostrando información y la fecha de la transacción.

Extensiones: Transferencias de fondos entre cuentas del usuario; obtención del menú de Cuentas.

Usos: Login / Autentificación.

CU05B – Transferencia fallida de fondos entre cuentas del usuario.

Definición: El usuario escoge la opción de Transferencia de Fondos. El sistema responde con la pantalla de Transferencia de Fondos. El usuario selecciona la cuenta origen. El usuario selecciona la cuenta destino de los fondos. El usuario ingresa el monto de la transferencia. El Usuario ingresa la clave de la transacción. El usuario sumite los datos. El sistema responde con la pantalla de Fondos Transferidos mostrando un mensaje de error.

Extensiones: Transferencias de fondos entre cuentas del usuario;

Usos: Login / Autentificación; obtención del menú de Cuentas.

CU06 – Cambio de claves.

Definición: El usuario escoge la opción de Usuario. El sistema responde con la pantalla de Usuario. El usuario selecciona la opción de Cambio de claves. El sistema responde con la pantalla de Cambio de Clave. El usuario selecciona qué clave desea cambiar. El usuario ingresa la clave anterior, la nueva y la confirmación de la nueva clave. El usuario sumite los cambios. El sistema responde.

Extensiones: Ninguna.

Usos: Login / Autentificación.

CU06A – Cambio exitoso de claves.

Definición: El usuario escoge la opción de Usuario. El sistema responde con la pantalla de Usuario. El usuario selecciona la opción de Cambio de claves. El sistema responde con la pantalla de Cambio de Clave. El usuario selecciona qué clave desea cambiar. El usuario ingresa la clave anterior, la nueva y la confirmación de la nueva clave. El usuario sumite los cambios. El sistema responde con la pantalla de Cuentas.

Extensiones: Cambiar claves.

Usos: Login / Autentificación.

CU06B – Cambio fallido de claves.

Definición: El usuario escoge la opción de Usuario. El sistema responde con la pantalla de Usuario. El usuario selecciona la opción de Cambio de claves. El sistema responde con la pantalla de Cambio de Clave. El usuario selecciona qué clave desea cambiar. El usuario ingresa la clave anterior, la nueva y la confirmación de la nueva clave. El usuario sumite los cambios. El sistema responde con la pantalla de Cambio de Claves mostrando un mensaje de error.

Extensiones: Cambio de claves.

Usos: Login / Autentificación.

CU07 – Logout.

Definición: El usuario selecciona la opción de Logout. El sistema desconecta al usuario y responde con la pantalla de Logout.

Extensiones: Ninguna.

Usos: Login / Autentificación.

Anexo B: Código fuente de la Aplicación Bancaria.

Espol AccountServlet.java:

```
package IDLDevelopment;

/**  This type was created in VisualAge. */
public class EspolAccountServlet extends com.ibm.jbroker.GenericObjectServlet {
	private _ORBAccountSkeleton ivj_ORBAccountSkeleton1 = null;
	private com.ibm.CORBA.iiop.ORB ivjORB1 = null;
	private SimpleAccount ivjSimpleAccount1 = null;
/**
 * Constructor
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public EspolAccountServlet() {
	super();
	initialize();
}
/**
 * connEtoM1:  (EspolAccountServlet.initialize() --> _ORBAccountSkeleton1.setImpl(Ljava.lang.Object;)V)
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM1() {
	try {
		// user code begin {1}
		// user code end
		get_ORBAccountSkeleton1().setImpl(getSimpleAccount1());
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {3}
		// user code end
		handleException(ivjExc);
	}
}
/**
 * connEtoM2:  (EspolAccountServlet.initialize() --> ORB1.connect(Lorg.omg.CORBA.Object;)V)
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM2() {
	try {
		// user code begin {1}
		// user code end
		getORB1().connect(get_ORBAccountSkeleton1());
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {3}
		// user code end
		handleException(ivjExc);
	}
}
/**
 * Return the _ORBAccountSkeleton1 property value. @return IDLDevelopment._ORBAccountSkeleton
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private _ORBAccountSkeleton get_ORBAccountSkeleton1() {
	if (ivj_ORBAccountSkeleton1 == null) {
		try {
			ivj_ORBAccountSkeleton1 = new IDLDevelopment._ORBAccountSkeleton();
			// user code begin {1}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {2}
			// user code end
			handleException(ivjExc);
		}
	};
	return ivj_ORBAccountSkeleton1;
}
/**
```

```
 * This method was created in VisualAge. @return java.util.Properties
 */
public java.util.Properties getImpls() {
	java.util.Properties p = new java.util.Properties();
	try {
		p.put("IDLDevelopment.ORBAccount", "IDLDevelopment.SimpleAccount");
	}catch (Exception e){
		e.printStackTrace();
	}
	return p;
}
/**
 * Return the ORB1 property value. @return com.ibm.CORBA.iiop.ORB
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private com.ibm.CORBA.iiop.ORB getORB1() {
	if (ivjORB1 == null) {
		try {
			ivjORB1 = new com.ibm.CORBA.iiop.ORB();
			// user code begin {1}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {2}
			// user code end
			handleException(ivjExc);
		}
	};
	return ivjORB1;
}
/**
 * Return the SimpleAccount1 property value. @return IDLDevelopment.SimpleAccount
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private SimpleAccount getSimpleAccount1() {
	if (ivjSimpleAccount1 == null) {
		try {
			ivjSimpleAccount1 = new IDLDevelopment.SimpleAccount();
			// user code begin {1}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {2}
			// user code end
			handleException(ivjExc);
		}
	};
	return ivjSimpleAccount1;
}
/**
 * Called whenever the part throws an exception. @param exception java.lang.Throwable
 */
private void handleException(Throwable exception) {

	/* Uncomment the following lines to print uncaught exceptions to stdout */
	System.out.println("--------- UNCAUGHT EXCEPTION ---------");
	exception.printStackTrace(System.out);
}
/**
 * Initializes connections
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void initConnections() {
	// user code begin {1}
	// user code end
}
/**
 * Initialize the class.
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void initialize() {
	// user code begin {1}
	// user code end
	initConnections();
	connEtoM1();
	connEtoM2();
	// user code begin {2}
```

```
		// user code end
}
/**
 * main entrypoint - starts the part when it is run as an application. @param args java.lang.String[]
 */
public static void main(java.lang.String[] args) {
		try {
			EspolAccountServlet aEspolAccountServlet;
			aEspolAccountServlet = new EspolAccountServlet();
		} catch (Throwable exception) {
			System.err.println("Exception occurred in main() of com.ibm.jbroker.GenericObjectServlet");
			exception.printStackTrace(System.out);
		}
}
}
```

EspolBankServlet.java:

```
package IDLDevelopment;

/**
 * This type was created in VisualAge.
 */
public class EspolBankServlet extends com.ibm.jbroker.GenericObjectServlet {
		private _ORBAccountSkeleton ivj_ORBAccountSkeleton1 = null;
		private com.ibm.CORBA.iiop.ORB ivjORB1 = null;
		private SimpleAccount ivjSimpleAccount1 = null;
/**
 * Constructor
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public EspolBankServlet() {
		super();
		initialize();
}
/**
 * connEtoM1:  (EspolBankServlet.initialize() --> _ORBAccountSkeleton1.setImpl(Ljava.lang.Object;)V)
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM1() {
		try {
			// user code begin {1}
			// user code end
			get_ORBAccountSkeleton1().setImpl(getSimpleAccount1());
			// user code begin {2}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {3}
			// user code end
			handleException(ivjExc);
		}
}
/**
 * connEtoM2:  (EspolBankServlet.initialize() --> ORB1.connect(Lorg.omg.CORBA.Object;)V)
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM2() {
		try {
			// user code begin {1}
			// user code end
			getORB1().connect(get_ORBAccountSkeleton1());
			// user code begin {2}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {3}
			// user code end
			handleException(ivjExc);
		}
}
/**
 * Return the _ORBAccountSkeleton1 property value. @return IDLDevelopment._ORBAccountSkeleton
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private _ORBAccountSkeleton get_ORBAccountSkeleton1() {
```

```
		if (ivj_ORBAccountSkeleton1 == null) {
			try {
				ivj_ORBAccountSkeleton1 = new IDLDevelopment._ORBAccountSkeleton();
				// user code begin {1}
				// user code end
			} catch (java.lang.Throwable ivjExc) {
				// user code begin {2}
				// user code end
				handleException(ivjExc);
			}
		};
		return ivj_ORBAccountSkeleton1;
}
/**
 * This method was created in VisualAge.  @return java.util.Properties
 */
public java.util.Properties getImpls() {
		java.util.Properties p = new java.util.Properties();
		try {
			p.put("IDLDevelopment.ORBAccount", "IDLDevelopment.SimpleAccount");
		}catch (Exception e){
			e.printStackTrace();
		}
		return p;
}
/**
 * Return the ORB1 property value. @return com.ibm.CORBA.iiop.ORB
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private com.ibm.CORBA.iiop.ORB getORB1() {
		if (ivjORB1 == null) {
			try {
				ivjORB1 = new com.ibm.CORBA.iiop.ORB();
				// user code begin {1}
				// user code end
			} catch (java.lang.Throwable ivjExc) {
				// user code begin {2}
				// user code end
				handleException(ivjExc);
			}
		};
		return ivjORB1;
}
/**
 * Return the SimpleAccount1 property value. @return IDLDevelopment.SimpleAccount
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private SimpleAccount getSimpleAccount1() {
		if (ivjSimpleAccount1 == null) {
			try {
				ivjSimpleAccount1 = new IDLDevelopment.SimpleAccount();
				// user code begin {1}
				// user code end
			} catch (java.lang.Throwable ivjExc) {
				// user code begin {2}
				// user code end
				handleException(ivjExc);
			}
		};
		return ivjSimpleAccount1;
}
/**
 * Called whenever the part throws an exception. @param exception java.lang.Throwable
 */
private void handleException(Throwable exception) {

		/* Uncomment the following lines to print uncaught exceptions to stdout */
		System.out.println("--------- UNCAUGHT EXCEPTION ---------");
		exception.printStackTrace(System.out);
}
/**
 * Initializes connections
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void initConnections() {
```

```
		// user code begin {1}
		// user code end
}
/**
 * Initialize the class.
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void initialize() {
		// user code begin {1}
		// user code end
		initConnections();
		connEtoM1();
		connEtoM2();
		// user code begin {2}
		// user code end
}
/**
 * main entrypoint - starts the part when it is run as an application.  @param args java.lang.String[]
 */
public static void main(java.lang.String[] args) {
		try {
			EspolBankServlet aEspolBankServlet;
			aEspolBankServlet = new EspolBankServlet();
		} catch (Throwable exception) {
			System.err.println("Exception occurred in main() of com.ibm.jbroker.GenericObjectServlet");
			exception.printStackTrace(System.out);
		}
}
}
```

EspolClient.java:

```
package IDLDevelopment;

import java.applet.*;
import java.awt.*;
/**
 * This type was created in VisualAge.
 */
public class EspolClient extends Applet implements java.awt.event.ActionListener {
		private Button ivjButton1 = null;
		private Button ivjButton2 = null;
		private TextField ivjTextField1 = null;
		private TextField ivjTextField2 = null;
		private Label ivjLabel1 = null;
		private Label ivjLabel2 = null;
		private Label ivjLabel3 = null;
		private Label ivjLabel4 = null;
		private com.ibm.CORBA.iiop.ORB orb = null;
		private com.ibm.jbroker.IHome theHome = null;
		private ORBAccount ivjORBAccount1 = null;
/**
 * Method to handle events for the ActionListener interface.  @param e java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public void actionPerformed(java.awt.event.ActionEvent e) {
		// user code begin {1}
		// user code end
		if ((e.getSource() == getButton1()) ) {
			connEtoM1(e);
		}
		if ((e.getSource() == getButton2()) ) {
			connEtoM2(e);
		}
		// user code begin {2}
		// user code end
}
/**
 * connEtoM1:  (Button1.action.actionPerformed(java.awt.event.ActionEvent) -->
SimpleAccount1.deposit(FLjava.lang.String;)F)
 * @return float @param arg1 java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private float connEtoM1(java.awt.event.ActionEvent arg1) {
```

```
		float connEtoM1Result = 0;
		try {
			// user code begin {1}
			// user code end
			connEtoM1Result = getORBAccount1().deposit(new Float(getTextField1().getText()).floatValue(),
getTextField2().getText());
			connEtoM4(connEtoM1Result);
			// user code begin {2}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {3}
			// user code end
			handleException(ivjExc);
		}
		return connEtoM1Result;
}
/**
 * connEtoM2:  (Button2.action.actionPerformed(java.awt.event.ActionEvent) -->
SimpleAccount1.withdraw(FLjava.lang.String;)F)
 * @return float  @param arg1 java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private float connEtoM2(java.awt.event.ActionEvent arg1) {
		float connEtoM2Result = 0;
		try {
			// user code begin {1}
			// user code end
			connEtoM2Result = getORBAccount1().withdraw(new
Float(getTextField1().getText()).floatValue(), getTextField2().getText());
			connEtoM3(connEtoM2Result);
			// user code begin {2}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {3}
			// user code end
			handleException(ivjExc);
		}
		return connEtoM2Result;
}
/**
 * connEtoM3:  ( (Button2,action.actionPerformed(java.awt.event.ActionEvent) -->
ORBAccount1,withdraw(FLjava.lang.String;)F).normalResult --> Label1.text)
 * @param result float
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM3(float result) {
		try {
			// user code begin {1}
			// user code end
			getLabel1().setText(String.valueOf(result));
			// user code begin {2}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {3}
			// user code end
			handleException(ivjExc);
		}
}
/**
 * connEtoM4:  ( (Button1,action.actionPerformed(java.awt.event.ActionEvent) -->
SimpleAccount1,deposit(FLjava.lang.String;)F).normalResult --> Label1.text)  @param result float
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM4(float result) {
		try {
			// user code begin {1}
			// user code end
			getLabel1().setText(String.valueOf(result));
			// user code begin {2}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {3}
			// user code end
			handleException(ivjExc);
		}
```

```
}
/**
 * Gets the applet information. @return java.lang.String
 */
public String getAppletInfo() {
	return "IDLDevelopment.EspolClient created using VisualAge for Java.";
}

/**
 * Return the ORBAccount1 property value.  @return IDLDevelopment.ORBAccount
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private ORBAccount getORBAccount1() {
	// user code begin {1}
	// user code end
	return ivjORBAccount1;
}

/**
 * Called whenever the part throws an exception. @param exception java.lang.Throwable
 */
private void handleException(Throwable exception) {

	/* Uncomment the following lines to print uncaught exceptions to stdout */
	System.out.println("--------- UNCAUGHT EXCEPTION ---------");
	exception.printStackTrace(System.out);
}
/**
 * Handle the Applet init method.
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public void init() {
	super.init();
	try {
		setName("EspolClient");
		setLayout(null);
		setSize(500, 400);
		add(getButton1(), getButton1().getName());
		add(getButton2(), getButton2().getName());
		add(getTextField1(), getTextField1().getName());
		add(getTextField2(), getTextField2().getName());
		add(getLabel1(), getLabel1().getName());
		add(getLabel2(), getLabel2().getName());
		add(getLabel3(), getLabel3().getName());
		add(getLabel4(), getLabel4().getName());
		initConnections();
		// user code begin {1}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {2}
		// user code end
		handleException(ivjExc);
	}
}
/**
 * Initializes connections
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void initConnections() {
	// user code begin {1}
	try{
		java.util.Properties props = new java.util.Properties();
		props.put("org.omg.CORBA.ORBClass","com.ibm.CORBA.iiop.ORB");
		orb = (com.ibm.CORBA.iiop.ORB) org.omg.CORBA.ORB.init((String[]) null,props);
		System.out.println("COMPLETED: ORB.init");
		try{
			java.net.URL url = new
java.net.URL("http://telemserver/servlet/IDLDevelopment.EspolBankServlet");
			theHome=com.ibm.jbroker.Utility.getIHome(orb,url);
			System.out.println("COMPLETED: Got theIHome: " + theHome);
		}catch (java.net.MalformedURLException e) {
			e.printStackTrace();
		}
		org.omg.CORBA.Object theObj = theHome.create("IDLDevelopment.ORBAccount");
		setORBAccount1(ORBAccountHelper.narrow(theObj));
```

```
                    if (getORBAccount1()!= null)
                    System.out.println("COMPLETED: Narrowed to the ORBAccount interface");
            }catch (org.omg.CORBA.SystemException e){
                    e.printStackTrace();
            }


            // user code end
            getButton1().addActionListener(this);
            getButton2().addActionListener(this);
}
/**
 * main entrypoint - starts the part when it is run as an application  @param args java.lang.String[]
 */
public static void main(java.lang.String[] args) {
            try {
                    Frame frame;
                    try {
                            Class aFrameClass = Class.forName("com.ibm.uvm.abt.edit.TestFrame");
                            frame = (Frame)aFrameClass.newInstance();
                    } catch (java.lang.Throwable ivjExc) {
                            frame = new Frame();
                    }
                    EspolClient aEspolClient;
                    Class iiCls = Class.forName("IDLDevelopment.EspolClient");
                    ClassLoader iiClsLoader = iiCls.getClassLoader();
                    aEspolClient =
(EspolClient)java.beans.Beans.instantiate(iiClsLoader,"IDLDevelopment.EspolClient");
                    frame.add("Center", aEspolClient);
                    frame.setSize(aEspolClient.getSize());
                    frame.setVisible(true);
            } catch (Throwable exception) {
                    System.err.println("Exception occurred in main() of java.applet.Applet");
                    exception.printStackTrace(System.out);
            }
}
/**
 * Set the ORBAccount1 to a new value.  @param newValue IDLDevelopment.ORBAccount
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void setORBAccount1(ORBAccount newValue) {
            if (ivjORBAccount1 != newValue) {
                    try {
                            ivjORBAccount1 = newValue;
                            // user code begin {1}
                            // user code end
                    } catch (java.lang.Throwable ivjExc) {
                            // user code begin {2}
                            // user code end
                            handleException(ivjExc);
                    }
            };
}
}
```

EspolCustomerServlet.java:

```
package IDLDevelopment;

/**
 * This type was created in VisualAge.
 */
public class EspolCustomerServlet extends com.ibm.jbroker.GenericObjectServlet {
            private _ORBCustomerSkeleton ivj_ORBCustomerSkeleton1 = null;
            private com.ibm.CORBA.iiop.ORB ivjORB1 = null;
            private SimpleCustomer ivjSimpleCustomer1 = null;
/**
 * Constructor
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public EspolCustomerServlet() {
            super();
            initialize();
}
```

```
/**
 * connEtoM1:  (EspolCustomerServlet.initialize() --> _ORBCustomerSkeleton1.setImpl(Ljava.lang.Object;)V)
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM1() {
	try {
		// user code begin {1}
		// user code end
		get_ORBCustomerSkeleton1().setImpl(getSimpleCustomer1());
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {3}
		// user code end
		handleException(ivjExc);
	}
}
/**
 * connEtoM2:  (EspolCustomerServlet.initialize() --> ORB1.connect(Lorg.omg.CORBA.Object;)V)
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM2() {
	try {
		// user code begin {1}
		// user code end
		getORB1().connect(get_ORBCustomerSkeleton1());
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {3}
		// user code end
		handleException(ivjExc);
	}
}
/**
 * Return the _ORBCustomerSkeleton1 property value.  @return IDLDevelopment._ORBCustomerSkeleton
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private _ORBCustomerSkeleton get_ORBCustomerSkeleton1() {
	if (ivj_ORBCustomerSkeleton1 == null) {
		try {
			ivj_ORBCustomerSkeleton1 = new IDLDevelopment._ORBCustomerSkeleton();
			// user code begin {1}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {2}
			// user code end
			handleException(ivjExc);
		}
	};
	return ivj_ORBCustomerSkeleton1;
}
/**
 * This method was created in VisualAge.  @return java.util.Properties
 */
public java.util.Properties getImpls() {
	java.util.Properties p = new java.util.Properties();
	try {
		p.put("IDLDevelopment.ORBCustomer", "IDLDevelopment.SimpleCustomer");
	}catch (Exception e){
		e.printStackTrace();
	}
	return p;
}
/**
 * Return the ORB1 property value.  @return com.ibm.CORBA.iiop.ORB
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private com.ibm.CORBA.iiop.ORB getORB1() {
	if (ivjORB1 == null) {
		try {
			ivjORB1 = new com.ibm.CORBA.iiop.ORB();
			// user code begin {1}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
```

```
                                        // user code begin {2}
                                        // user code end
                                        handleException(ivjExc);
                                }
                        };
                        return ivjORB1;
        }
        /**
         * Return the SimpleCustomer1 property value. @return IDLDevelopment.SimpleCustomer
         */
        /* WARNING: THIS METHOD WILL BE REGENERATED. */
        private SimpleCustomer getSimpleCustomer1() {
                        if (ivjSimpleCustomer1 == null) {
                                try {
                                        ivjSimpleCustomer1 = new IDLDevelopment.SimpleCustomer();
                                        // user code begin {1}
                                        // user code end
                                } catch (java.lang.Throwable ivjExc) {
                                        // user code begin {2}
                                        // user code end
                                        handleException(ivjExc);
                                }
                        };
                        return ivjSimpleCustomer1;
        }
        /**
         * Called whenever the part throws an exception. @param exception java.lang.Throwable
         */
        private void handleException(Throwable exception) {

                        /* Uncomment the following lines to print uncaught exceptions to stdout */
                        System.out.println("--------- UNCAUGHT EXCEPTION ---------");
                        exception.printStackTrace(System.out);
        }
        /**
         * Initializes connections
         */
        /* WARNING: THIS METHOD WILL BE REGENERATED. */
        private void initConnections() {
                        // user code begin {1}
                        // user code end
        }
        /**
         * Initialize the class.
         */
        /* WARNING: THIS METHOD WILL BE REGENERATED. */
        private void initialize() {
                        // user code begin {1}
                        // user code end
                        initConnections();
                        connEtoM1();
                        connEtoM2();
                        // user code begin {2}
                        // user code end
        }
        /**
         * main entrypoint - starts the part when it is run as an application. @param args java.lang.String[]
         */
        public static void main(java.lang.String[] args) {
                        try {
                                EspolCustomerServlet aEspolCustomerServlet;
                                aEspolCustomerServlet = new EspolCustomerServlet();
                        } catch (Throwable exception) {
                                System.err.println("Exception occurred in main() of com.ibm.jbroker.GenericObjectServlet");
                                exception.printStackTrace(System.out);
                        }
        }
        }
```

EspolTransServlet.java:

```
package IDLDevelopment;

/**
```

```
 * This type was created in VisualAge.
 */
public class EspolTransServlet extends com.ibm.jbroker.GenericObjectServlet {
	private _ORBTransSkeleton ivj_ORBTransSkeleton1 = null;
	private com.ibm.CORBA.iiop.ORB ivjORB1 = null;
	private SimpleTrans ivjSimpleTrans1 = null;
/**
 * Constructor
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public EspolTransServlet() {
	super();
	initialize();
}
/**
 * connEtoM1:  (EspolTransServlet.initialize() --> _ORBTransSkeleton1.setImpl(Ljava.lang.Object;)V)
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM1() {
	try {
		// user code begin {1}
		// user code end
		get_ORBTransSkeleton1().setImpl(getSimpleTrans1());
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {3}
		// user code end
		handleException(ivjExc);
	}
}
/**
 * connEtoM2:  (EspolTransServlet.initialize() --> ORB1.connect(Lorg.omg.CORBA.Object;)V)
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM2() {
	try {
		// user code begin {1}
		// user code end
		getORB1().connect(get_ORBTransSkeleton1());
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {3}
		// user code end
		handleException(ivjExc);
	}
}
/**
 * Return the _ORBTransSkeleton1 property value.  @return IDLDevelopment._ORBTransSkeleton
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private _ORBTransSkeleton get_ORBTransSkeleton1() {
	if (ivj_ORBTransSkeleton1 == null) {
		try {
			ivj_ORBTransSkeleton1 = new IDLDevelopment._ORBTransSkeleton();
			// user code begin {1}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {2}
			// user code end
			handleException(ivjExc);
		}
	};
	return ivj_ORBTransSkeleton1;
}
/**
 * This method was created in VisualAge.  @return java.util.Properties
 */
public java.util.Properties getImpls() {
	java.util.Properties p = new java.util.Properties();
	try {
		p.put("IDLDevelopment.ORBTrans", "IDLDevelopment.SimpleTrans");
	}catch (Exception e){
		e.printStackTrace();
```

```
		}
		return p;
}
/**
 * Return the ORB1 property value. @return com.ibm.CORBA.iiop.ORB
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private com.ibm.CORBA.iiop.ORB getORB1() {
		if (ivjORB1 == null) {
			try {
				ivjORB1 = new com.ibm.CORBA.iiop.ORB();
				// user code begin {1}
				// user code end
			} catch (java.lang.Throwable ivjExc) {
				// user code begin {2}
				// user code end
				handleException(ivjExc);
			}
		};
		return ivjORB1;
}
/**
 * Return the SimpleTrans1 property value.  @return IDLDevelopment.SimpleTrans
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private SimpleTrans getSimpleTrans1() {
		if (ivjSimpleTrans1 == null) {
			try {
				ivjSimpleTrans1 = new IDLDevelopment.SimpleTrans();
				// user code begin {1}
				// user code end
			} catch (java.lang.Throwable ivjExc) {
				// user code begin {2}
				// user code end
				handleException(ivjExc);
			}
		};
		return ivjSimpleTrans1;
}
/**
 * Called whenever the part throws an exception.  @param exception java.lang.Throwable
 */
private void handleException(Throwable exception) {

		/* Uncomment the following lines to print uncaught exceptions to stdout */
		System.out.println("--------- UNCAUGHT EXCEPTION ---------");
		exception.printStackTrace(System.out);
}
/**
 * Initializes connections
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void initConnections() {
		// user code begin {1}
		// user code end
}
/**
 * Initialize the class.
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void initialize() {
		// user code begin {1}
		// user code end
		initConnections();
		connEtoM1();
		connEtoM2();
		// user code begin {2}
		// user code end
}
/**
 * main entrypoint - starts the part when it is run as an application. @param args java.lang.String[]
 */
public static void main(java.lang.String[] args) {
		try {
			EspolTransServlet aEspolTransServlet;
```

```
                    aEspolTransServlet = new EspolTransServlet();
            } catch (Throwable exception) {
                    System.err.println("Exception occurred in main() of com.ibm.jbroker.GenericObjectServlet");
                    exception.printStackTrace(System.out);
            }
    }
}
```

Login.java:

```
package IDLDevelopment;

import java.applet.*;
import java.awt.*;
import java.io.*;
import java.util.*;
import java.net.URL;
import java.net.MalformedURLException;
/**
 * This type was created in VisualAge.
 */
public class Login extends Applet implements java.awt.event.ActionListener {
            private Button ivjButton1 = null;
            private Button ivjButton2 = null;
            private TextField ivjTextField1 = null;
            private TextField ivjTextField2 = null;
            private Label ivjLabel1 = null;
            private Label ivjLabel2 = null;
            private Label ivjLabel3 = null;
            private com.ibm.CORBA.iiop.ORB orb = null;
            private com.ibm.jbroker.IHome theHome = null;
            private ORBCustomer ivjORBCustomer1 = null;
            private String customer = new String();
            private String customerId = new String();

/**
 * Method to handle events for the ActionListener interface.  @param e java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public void actionPerformed(java.awt.event.ActionEvent e) {
            // user code begin {1}
            // user code end
            if ((e.getSource() == getButton1()) ) {
                    connEtoM1(e);
            }
            if ((e.getSource() == getButton2()) ) {
                    connEtoM2(e);
            }
            // user code begin {2}
            // user code end
}
/**
 * connEtoM1:  (Button1.action.actionPerformed(java.awt.event.ActionEvent) -->
SimpleCustomer1.deposit(FLjava.lang.String;)F)
 * @return float  @param arg1 java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private boolean connEtoM1(java.awt.event.ActionEvent arg1) {
            boolean connEtoM1Result = false;
            try {
                    // user code begin {1}
                    // user code end
                    connEtoM1Result = getORBCustomer1().checkPassword(getTextField2().getText(),
getTextField1().getText());
                    connEtoM4(connEtoM1Result);
                    // user code begin {2}
                    // user code end
            } catch (java.lang.Throwable ivjExc) {
                    // user code begin {3}
                    // user code end
                    handleException(ivjExc);
            }
            return connEtoM1Result;
}
```

```
/**
 * connEtoM2:  (Button2.action.actionPerformed(java.awt.event.ActionEvent) -->
SimpleCustomer1.withdraw(FLjava.lang.String;)F)
 * @return float @param arg1 java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private float connEtoM2(java.awt.event.ActionEvent arg1) {
	float connEtoM2Result = 0;
	try {
		// user code begin {1}
		// user code end
		//connEtoM2Result = getORBCustomer1().withdraw(new
Float(getTextField1().getText()).floatValue(), getTextField2().getText());
		//connEtoM3(connEtoM2Result);
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {3}
		// user code end
		handleException(ivjExc);
	}
	return connEtoM2Result;
}
/**
 * connEtoM3:  ( (Button2,action.actionPerformed(java.awt.event.ActionEvent) -->
ORBCustomer1,withdraw(FLjava.lang.String;)F).normalResult --> Label1.text)
 * @param result float
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM3(float result) {
	try {
		// user code begin {1}
		// user code end
		getLabel1().setText(String.valueOf(result));
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {3}
		// user code end
		handleException(ivjExc);
	}
}
/**
 * connEtoM4:  ( (Button1,action.actionPerformed(java.awt.event.ActionEvent) -->
SimpleCustomer1,deposit(FLjava.lang.String;)F).normalResult --> Label1.text)
 * @param result float
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM4(boolean result) {
	AppletContext appletContext = getAppletContext();
	try {
		// user code begin {1}
		// user code end
		if (result) {
			customer = getORBCustomer1().getGrettings(getTextField2().getText());
			customerId = getTextField2().getText();

			getLabel1().setText("Transaccion OK");

			//String urlString = new String(getCodeBase()+"bank.html");
			String urlString = new
String("http://"+getCodeBase().getHost()+"/servlet/IDLDevelopment.BankAdminServlet?option=login&userid="+custo
merId);
			URL url = null;
			try {
				url = new URL(urlString);
			} catch (MalformedURLException e) {
			System.out.println("Malformed URL: " + urlString);
			}
			if (url != null) {
				appletContext.showDocument(url, "_self");
			}

		} else {
			getLabel1().setText("Problemas con al procesar login");
```

```
                }
                // user code begin {2}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {3}
                // user code end
                handleException(ivjExc);
        }
}
/**
 * Gets the applet information. @return java.lang.String
 */
public String getAppletInfo() {
        return "IDLDevelopment.Login created using VisualAge for Java.";
}
/**
 * Return the ORBCustomer1 property value.  @return IDLDevelopment.ORBCustomer
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private ORBCustomer getORBCustomer1() {
        // user code begin {1}
        // user code end
        return ivjORBCustomer1;
}
/**
 * Called whenever the part throws an exception. @param exception java.lang.Throwable
 */
private void handleException(Throwable exception) {

        /* Uncomment the following lines to print uncaught exceptions to stdout */
        System.out.println("--------- UNCAUGHT EXCEPTION ---------");
        exception.printStackTrace(System.out);
}
/**
 * Handle the Applet init method.
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public void init() {
        super.init();
        try {
                setName("Login");
                setLayout(null);
                setSize(300, 185);
                add(getButton1(), getButton1().getName());
                add(getButton2(), getButton2().getName());
                add(getTextField2(), getTextField2().getName());
                add(getTextField1(), getTextField1().getName());
                add(getLabel1(), getLabel1().getName());
                add(getLabel2(), getLabel2().getName());
                add(getLabel3(), getLabel3().getName());
                initConnections();
                // user code begin {1}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {2}
                // user code end
                handleException(ivjExc);
        }
}
/**
 * Initializes connections
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void initConnections() {
        // user code begin {1}
        try{
                java.util.Properties props = new java.util.Properties();
                props.put("org.omg.CORBA.ORBClass","com.ibm.CORBA.iiop.ORB");
                orb = (com.ibm.CORBA.iiop.ORB) org.omg.CORBA.ORB.init((String[]) null,props);
                System.out.println("COMPLETED: ORB.init");
                try{
                        java.net.URL url = new
java.net.URL("http://"+getCodeBase().getHost()+"/servlet/IDLDevelopment.EspolCustomerServlet");
                        theHome=com.ibm.jbroker.Utility.getIlHome(orb,url);
                        System.out.println("COMPLETED: Got theIlHome: " + theHome);
```

```
                }catch (java.net.MalformedURLException e) {
                        e.printStackTrace();
                }
                org.omg.CORBA.Object theObj = theHome.create("IDLDevelopment.ORBCustomer");
                setORBCustomer1(ORBCustomerHelper.narrow(theObj));
                if (getORBCustomer1()!= null)
                System.out.println("COMPLETED: Narrowed to the ORBCustomer interface");
        }catch (org.omg.CORBA.SystemException e){
                e.printStackTrace();
        }


        // user code end
        getButton1().addActionListener(this);
        getButton2().addActionListener(this);
}
/**
 * main entrypoint - starts the part when it is run as an application  @param args java.lang.String[]
 */
public static void main(java.lang.String[] args) {
        try {
                Frame frame;
                try {
                        Class aFrameClass = Class.forName("com.ibm.uvm.abt.edit.TestFrame");
                        frame = (Frame)aFrameClass.newInstance();
                } catch (java.lang.Throwable ivjExc) {
                        frame = new Frame();
                }
                Login aLogin = new Login();
                //Class iiCls = Class.forName("IDLDevelopment.Login");
                //ClassLoader iiClsLoader = iiCls.getClassLoader();
                //aLogin = (Login)java.beans.Beans.instantiate(iiClsLoader,"IDLDevelopment.Login");
                frame.add("Center", aLogin);
                frame.setSize(aLogin.getSize());
                frame.setVisible(true);
        } catch (Throwable exception) {
                System.err.println("Exception occurred in main() of java.applet.Applet");
                exception.printStackTrace(System.out);
        }
}
/**
 * Set the ORBCustomer1 to a new value.
 * @param newValue IDLDevelopment.ORBCustomer
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void setORBCustomer1(ORBCustomer newValue) {
        if (ivjORBCustomer1 != newValue) {
                try {
                        ivjORBCustomer1 = newValue;
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        };
}
}
```

NewAccount.java:

```
package IDLDevelopment;

import java.applet.*;
import java.awt.*;
/**
 * This type was created in VisualAge.
 */
public class NewAccount extends Applet implements java.awt.event.ActionListener, java.awt.event.ItemListener {
        private Button ivjButton1 = null;
        private Button ivjButton2 = null;
        private Label ivjLabel1 = null;
        private TextField ivjTextField1 = null;
```

```
	private TextField ivjTextField2 = null;
	private com.ibm.CORBA.iiop.ORB orb = null;
	private com.ibm.jbroker.IHome theHome = null;
	private ORBAccount ivjORBAccount1 = null;
	//private SimpleAccount ivjORBAccount1 = null;
	private Label ivjLabel3 = null;
	private Label ivjLabel4 = null;
	String account = new String();
	String customerId = new String();
	String s = new String();
	private Choice ivjChoice1 = null;
	private Label ivjLabel2 = null;
/**
 * Method to handle events for the ActionListener interface.  @param e java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public void actionPerformed(java.awt.event.ActionEvent e) {
	// user code begin {1}
	// user code end
	if ((e.getSource() == getButton1()) ) {
		connEtoM1(e);
	}
	if ((e.getSource() == getButton2()) ) {
		connEtoM2(e);
	}
	// user code begin {2}
	// user code end
}
/**
 * connEtoC1:  (List1.item. --> NewAccount.list1_ItemEvent(Ljava.lang.String;)Ljava.lang.String;)
 * @return java.lang.String
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private String connEtoC1() {
	String connEtoC1Result = null;
	try {
		// user code begin {1}
		// user code end
		connEtoC1Result = this.list1_ItemEvent(getChoice1().getSelectedItem());
		connEtoM3(connEtoC1Result);
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {3}
		// user code end
		handleException(ivjExc);
	}
	return connEtoC1Result;
}
/**
 * connEtoM1:  (Button1.action.actionPerformed(java.awt.event.ActionEvent) -->
ORBAccount1.deposit(FLjava.lang.String;)F)
 * @return float @param arg1 java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private boolean connEtoM1(java.awt.event.ActionEvent arg1) {
	boolean connEtoM1Result = false;
	try {
		// user code begin {1}
		// user code end
		connEtoM1Result = getORBAccount1().createAccount("", customerId, getTextField2().getText(),
new Float(getTextField1().getText()).floatValue());
		connEtoM4(connEtoM1Result);
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {3}
		// user code end
		handleException(ivjExc);
	}
	return connEtoM1Result;
}
/**
 * connEtoM2:  (Button2.action.actionPerformed(java.awt.event.ActionEvent) -->
ORBAccount1.withdraw(FLjava.lang.String;)F)
```

```
 * @return float  @param arg1 java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private boolean connEtoM2(java.awt.event.ActionEvent arg1) {
	boolean connEtoM2Result = false;
	try {
		// user code begin {1}
		// user code end
		//connEtoM2Result = getORBAccount1().withdraw(new
Float(getTextField1().getText()).floatValue(), getTextField2().getText());
		//connEtoM5(connEtoM2Result);
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {3}
		// user code end
		handleException(ivjExc);
	}
	return connEtoM2Result;
}
/**
 * connEtoM3:  ( (List1,item. --> NewAccount,list1_ItemEvent(Ljava.lang.String;)Ljava.lang.String;).normalResult -->
TextField2.text)
 * @param result java.lang.String
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM3(String result) {
	try {
		// user code begin {1}
		// user code end
		getTextField2().setText(result);
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {3}
		// user code end
		handleException(ivjExc);
	}
}
/**
 * connEtoM4:  ( (Button1,action.actionPerformed(java.awt.event.ActionEvent) -->
ORBAccount1,deposit(FLjava.lang.String;)F).normalResult --> Label4.text)  @param result float
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM4(boolean result) {
	try {
		// user code begin {1}
		// user code end
		if (result) {
			getLabel4().setText("Transaccion OK");
		} else {
			getLabel4().setText("Problemas al crear cuenta");
		}
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {3}
		// user code end
		handleException(ivjExc);
	}
}
/**
 * connEtoM5:  ( (Button2,action.actionPerformed(java.awt.event.ActionEvent) -->
ORBAccount1,withdraw(FLjava.lang.String;)F).normalResult --> Label4.text)
 * @param result float
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM5(boolean result) {
	try {
		// user code begin {1}
		// user code end
		getLabel4().setText("");
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
```

```
                        // user code begin {3}
                        // user code end
                        handleException(ivjExc);
                }
}

/**
 * Gets the applet information.  @return java.lang.String
 */
public String getAppletInfo() {
                return "IDLDevelopment.NewAccount created using VisualAge for Java.";
}
/**
 * Return the Choice1 property value.  @return java.awt.Choice
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private Choice getChoice1() {
                if (ivjChoice1 == null) {
                        try {
                                ivjChoice1 = new java.awt.Choice();
                                ivjChoice1.setName("Choice1");
                                ivjChoice1.setBounds(175, 50, 137, 28);
                                // user code begin {1}
                                // user code end
                        } catch (java.lang.Throwable ivjExc) {
                                // user code begin {2}
                                // user code end
                                handleException(ivjExc);
                        }
                };
                return ivjChoice1;
}

/**
 * Return the ORBAccount1 property value.  @return IDLDevelopment.ORBAccount
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private ORBAccount getORBAccount1() {
//private SimpleAccount getORBAccount1() {
                // user code begin {1}
                // user code end
                return ivjORBAccount1;
}
/**
 * Called whenever the part throws an exception.  @param exception java.lang.Throwable
 */
private void handleException(Throwable exception) {

                /* Uncomment the following lines to print uncaught exceptions to stdout */
                // System.out.println("--------- UNCAUGHT EXCEPTION ---------");
                // exception.printStackTrace(System.out);
}
/**
 * Handle the Applet init method.
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public void init() {
                super.init();
                try {
                        setName("NewAccount");
                        setLayout(null);
                        setSize(370, 240);
                        add(getTextField1(), getTextField1().getName());
                        add(getButton1(), getButton1().getName());
                        add(getButton2(), getButton2().getName());
                        add(getLabel1(), getLabel1().getName());
                        add(getLabel3(), getLabel3().getName());
                        add(getLabel4(), getLabel4().getName());
                        add(getChoice1(), getChoice1().getName());
                        add(getLabel2(), getLabel2().getName());
                        initConnections();
                        // user code begin {1}

                        // user code end
                } catch (java.lang.Throwable ivjExc) {
```

```
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
}
/**
 * Initializes connections
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void initConnections() {
                // user code begin {1}

                try{
                        java.util.Properties props = new java.util.Properties();
                        props.put("org.omg.CORBA.ORBClass","com.ibm.CORBA.iiop.ORB");
                        orb = (com.ibm.CORBA.iiop.ORB) org.omg.CORBA.ORB.init((String[]) null,props);
                        System.out.println("COMPLETED: ORB.init");
                        try{
                                java.net.URL url = new
java.net.URL("http://"+getCodeBase().getHost()+"/servlet/IDLDevelopment.EspolAccountServlet");
                                theHome=com.ibm.jbroker.Utility.getIHome(orb,url);
                                System.out.println("COMPLETED: Got theIHome: " + theHome);
                        }catch (java.net.MalformedURLException e) {
                                e.printStackTrace();
                        }
                        org.omg.CORBA.Object theObj = theHome.create("IDLDevelopment.ORBAccount");
                        setORBAccount1(ORBAccountHelper.narrow(theObj));
                        if (getORBAccount1()!= null)
                        System.out.println("COMPLETED: Narrowed to the ORBAccount interface");
                }catch (org.omg.CORBA.SystemException e){
                        e.printStackTrace();
                }

                // setORBAccount1(new SimpleAccount());

                customerId = getParameter("customerId");
                if(customerId == null)
                        customerId = " ";
                getChoice1().add("CHK");
                getChoice1().add("SAV");
                getTextField2().setText("CHK");

                // user code end
                getChoice1().addItemListener(this);
                getButton1().addActionListener(this);
                getButton2().addActionListener(this);
}
/**
 * Method to handle events for the ItemListener interface.  @param e java.awt.event.ItemEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public void itemStateChanged(java.awt.event.ItemEvent e) {
                // user code begin {1}
                // user code end
                if ((e.getSource() == getChoice1()) ) {
                        connEtoC1();
                }
                // user code begin {2}
                // user code end
}
/**
 * Comment
 */
public String list1_ItemEvent(String s) {
                return s.trim();
}
/**
 * main entrypoint - starts the part when it is run as an application.  @param args java.lang.String[]
 */
public static void main(java.lang.String[] args) {
                try {
                        Frame frame;
                        try {
                                Class aFrameClass = Class.forName("com.ibm.uvm.abt.edit.TestFrame");
                                frame = (Frame)aFrameClass.newInstance();
```

```
                } catch (java.lang.Throwable ivjExc) {
                        frame = new Frame();
                }
                NewAccount aNewAccount = new NewAccount();
                frame.add("Center", aNewAccount);
                frame.setSize(aNewAccount.getSize());
                frame.setVisible(true);
        } catch (Throwable exception) {
                System.err.println("Exception occurred in main() of java.applet.Applet");
                exception.printStackTrace(System.out);
        }
}
/**
 * Set the ORBAccount1 to a new value.  @param newValue IDLDevelopment.ORBAccount
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void setORBAccount1(ORBAccount newValue) {
//private void setORBAccount1(SimpleAccount newValue) {
        if (ivjORBAccount1 != newValue) {
                try {
                        ivjORBAccount1 = newValue;
                        // user code begin {1}
                        // user code end
                } catch (java.lang.Throwable ivjExc) {
                        // user code begin {2}
                        // user code end
                        handleException(ivjExc);
                }
        };
}
}
```

NewUser.java:

```
package IDLDevelopment;

import java.applet.*;
import java.awt.*;
import java.io.*;
import java.util.*;
import java.net.URL;
import java.net.MalformedURLException;
/**
 * This type was created in VisualAge.
 */
public class NewUser extends Applet implements java.awt.event.ActionListener {
        private Button ivjButton1 = null;
        private Button ivjButton2 = null;
        private TextField ivjTextField1 = null;
        private TextField ivjTextField2 = null;
        private TextField ivjTextField3 = null;
        private TextField ivjTextField4 = null;
        private TextField ivjTextField5 = null;
        private Label ivjLabel1 = null;
        private Label ivjLabel2 = null;
        private Label ivjLabel3 = null;
        private Label ivjLabel4 = null;
        private Label ivjLabel5 = null;
        private Label ivjLabel6 = null;
        private com.ibm.CORBA.iiop.ORB orb = null;
        private com.ibm.jbroker.IHome theHome = null;
        private ORBCustomer ivjORBCustomer1 = null;
        private String customer = new String();
        private String customerId = new String();

/**
 * Method to handle events for the ActionListener interface.  @param e java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public void actionPerformed(java.awt.event.ActionEvent e) {
        // user code begin {1}
        // user code end
        if ((e.getSource() == getButton1()) ) {
                connEtoM1(e);
```

```
		}
		if ((e.getSource() == getButton2()) ) {
			connEtoM2(e);
		}
		// user code begin {2}
		// user code end
}
/**
 * connEtoM1:  (Button1.action.actionPerformed(java.awt.event.ActionEvent) -->
SimpleCustomer1.deposit(FLjava.lang.String;)F)
 * @return float  @param arg1 java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private boolean connEtoM1(java.awt.event.ActionEvent arg1) {
		boolean connEtoM1Result = false;
		try {
			// user code begin {1}
			// user code end
			connEtoM1Result = getORBCustomer1().createCustomer(getTextField1().getText(),
getTextField2().getText(), getTextField3().getText(), getTextField4().getText(),getTextField5().getText());
			connEtoM4(connEtoM1Result);
			// user code begin {2}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {3}
			// user code end
			handleException(ivjExc);
		}
		return connEtoM1Result;
}
/**
 * connEtoM2:  (Button2.action.actionPerformed(java.awt.event.ActionEvent) -->
SimpleCustomer1.withdraw(FLjava.lang.String;)F)
 * @return float  @param arg1 java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private float connEtoM2(java.awt.event.ActionEvent arg1) {
		float connEtoM2Result = 0;
		try {
			// user code begin {1}
			// user code end
			//connEtoM2Result = getORBCustomer1().withdraw(new
Float(getTextField1().getText()).floatValue(), getTextField2().getText());
			//connEtoM3(connEtoM2Result);
			// user code begin {2}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {3}
			// user code end
			handleException(ivjExc);
		}
		return connEtoM2Result;
}
/**
 * connEtoM3:  ( (Button2,action.actionPerformed(java.awt.event.ActionEvent) -->
ORBCustomer1,withdraw(FLjava.lang.String;)F).normalResult --> Label1.text)
 * @param result float
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM3(float result) {
		try {
			// user code begin {1}
			// user code end
			getLabel1().setText(String.valueOf(result));
			// user code begin {2}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {3}
			// user code end
			handleException(ivjExc);
		}
}
/**
 * connEtoM4:  ( (Button1,action.actionPerformed(java.awt.event.ActionEvent) -->
SimpleCustomer1,deposit(FLjava.lang.String;)F).normalResult --> Label1.text)
```

```
 * @param result float
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoM4(boolean result) {
	AppletContext appletContext = getAppletContext();
	try {
		// user code begin {1}
		// user code end
		if (result) {
			customer = getORBCustomer1().getGrettings(getTextField1().getText());
			customerId = getTextField1().getText();

			getLabel1().setText("Transaccion OK");

			//String urlString = new String(getCodeBase()+"bank.html");
			String urlString = new
String("http://"+getCodeBase().getHost()+"/servlet/IDLDevelopment.BankAdminServlet?option=login&userid="+custo
merId);
			URL url = null;
			try {
				url = new URL(urlString);
			} catch (MalformedURLException e) {
			System.out.println("Malformed URL: " + urlString);
			}
			if (url != null) {
				appletContext.showDocument(url, "_self");
			}

		} else {
			getLabel1().setText("Problemas al crear usuario");
		}
		// user code begin {2}
		// user code end
	} catch (java.lang.Throwable ivjExc) {
		// user code begin {3}
		// user code end
		handleException(ivjExc);
	}
}
/**
 * Gets the applet information. @return java.lang.String
 */
public String getAppletInfo() {
	return "IDLDevelopment.NewUser created using VisualAge for Java.";
}

/**
 * Return the ORBCustomer1 property value.  @return IDLDevelopment.ORBCustomer
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private ORBCustomer getORBCustomer1() {
	// user code begin {1}
	// user code end
	return ivjORBCustomer1;
}


/**
 * Called whenever the part throws an exception.  @param exception java.lang.Throwable
 */
private void handleException(Throwable exception) {

	/* Uncomment the following lines to print uncaught exceptions to stdout */
	System.out.println("--------- UNCAUGHT EXCEPTION ---------");
	exception.printStackTrace(System.out);
}
/**
 * Handle the Applet init method.
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public void init() {
	super.init();
	try {
		setName("NewUser");
		setLayout(null);
```

```
                setSize(350, 230);
                add(getButton1(), getButton1().getName());
                add(getButton2(), getButton2().getName());
                add(getTextField1(), getTextField1().getName());
                add(getTextField2(), getTextField2().getName());
                add(getTextField3(), getTextField3().getName());
                add(getTextField4(), getTextField4().getName());
                add(getTextField5(), getTextField5().getName());
                add(getLabel1(), getLabel1().getName());
                add(getLabel2(), getLabel2().getName());
                add(getLabel3(), getLabel3().getName());
                add(getLabel4(), getLabel4().getName());
                add(getLabel5(), getLabel5().getName());
                add(getLabel6(), getLabel6().getName());
                initConnections();
                // user code begin {1}
                // user code end
        } catch (java.lang.Throwable ivjExc) {
                // user code begin {2}
                // user code end
                handleException(ivjExc);
        }
}
/**
 * Initializes connections
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void initConnections() {

        // user code begin {1}
        try{
                java.util.Properties props = new java.util.Properties();
                props.put("org.omg.CORBA.ORBClass","com.ibm.CORBA.iiop.ORB");
                orb = (com.ibm.CORBA.iiop.ORB) org.omg.CORBA.ORB.init((String[]) null,props);
                System.out.println("COMPLETED: ORB.init");
                try{
                        java.net.URL url = new
java.net.URL("http://"+getCodeBase().getHost()+"/servlet/IDLDevelopment.EspolCustomerServlet");
                        theHome=com.ibm.jbroker.Utility.getIHome(orb,url);
                        System.out.println("COMPLETED: Got theIHome: " + theHome);
                }catch (java.net.MalformedURLException e) {
                        e.printStackTrace();
                }
                org.omg.CORBA.Object theObj = theHome.create("IDLDevelopment.ORBCustomer");
                setORBCustomer1(ORBCustomerHelper.narrow(theObj));
                if (getORBCustomer1()!= null)
                System.out.println("COMPLETED: Narrowed to the ORBCustomer interface");
        }catch (org.omg.CORBA.SystemException e){
                e.printStackTrace();
        }

        // user code end
        getButton1().addActionListener(this);
        getButton2().addActionListener(this);
}
/**
 * main entrypoint - starts the part when it is run as an application  @param args java.lang.String[]
 */
public static void main(java.lang.String[] args) {
        try {
                Frame frame;
                try {
                        Class aFrameClass = Class.forName("com.ibm.uvm.abt.edit.TestFrame");
                        frame = (Frame)aFrameClass.newInstance();
                } catch (java.lang.Throwable ivjExc) {
                        frame = new Frame();
                }
                NewUser aNewUser = new NewUser();
                //Class iiCls = Class.forName("IDLDevelopment.NewUser");
                //ClassLoader iiClsLoader = iiCls.getClassLoader();
                //aNewUser = (NewUser)java.beans.Beans.instantiate(iiClsLoader,"IDLDevelopment.NewUser");
                frame.add("Center", aNewUser);
                frame.setSize(aNewUser.getSize());
                frame.setVisible(true);
        } catch (Throwable exception) {
```

```
                        System.err.println("Exception occurred in main() of java.applet.Applet");
                        exception.printStackTrace(System.out);
                }
}
/**
 * Set the ORBCustomer1 to a new value.  @param newValue IDLDevelopment.ORBCustomer
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void setORBCustomer1(ORBCustomer newValue) {
                if (ivjORBCustomer1 != newValue) {
                        try {
                                ivjORBCustomer1 = newValue;
                                // user code begin {1}
                                // user code end
                        } catch (java.lang.Throwable ivjExc) {
                                // user code begin {2}
                                // user code end
                                handleException(ivjExc);
                        }
                };
}
}
```

OrbAccount.java:

```
package IDLDevelopment;

/**
* IDLDevelopment/ORBAccount.java
* Generated by the IBM IDL-to-Java compiler, version 1.0
* from acct.idl  Friday, July 30, 1999 11:51:06 o'clock AM EST
*/

public interface ORBAccount
{
  String fieldNumber ();
  void fieldNumber (String newFieldNumber);
  String fieldCustomerId ();
  void fieldCustomerId (String newFieldCustomerId);
  String fieldType ();
  void fieldType (String newFieldType);
  float fieldBalance ();
  void fieldBalance (float newFieldBalance);
  int accountCount ();
  void accountCount (int newAccountCount);
  float deposit (float howMuch, String acctnum);
  float withdraw (float howMuch, String acctnum);
  float transfer (float howMuch, String acctnum1, String acctnum2);
  float getBalance (String acctnum);
  void setBalance (float balance, String acctnum);
  void getAccountInfo (String customerId);
  boolean createAccount (String acctnum, String customerId, String type, float balance);
  boolean removeAccount (String acctnum);
  boolean findElement (String acctnum);
  void setAccountNumberArray (String s, int i);
  String getAccountNumberArray (int i);
  void setAccountTypeArray (String s, int i);
  String getAccountTypeArray (int i);
  void setBalanceArray (String s, int i);
  String getBalanceArray (int i);
  boolean saveTrans (String acctnum, String type, float amount);
} // interface ORBAccount
```

ORBCustomer,java:

```
package IDLDevelopment;

/**
* IDLDevelopment/ORBCustomer.java
* Generated by the IBM IDL-to-Java compiler, version 1.0
* from cust.idl  Wednesday, July 21, 1999 4:42:32 o'clock PM EST
*/
```

```
public interface ORBCustomer
{
  String fieldCustomerId ();
  void fieldCustomerId (String newFieldCustomerId);
  String fieldFirstName ();
  void fieldFirstName (String newFieldFirstName);
  String fieldLastName ();
  void fieldLastName (String newFieldLastName);
  String fieldTitle ();
  void fieldTitle (String newFieldTitle);
  String fieldPassword ();
  void fieldPassword (String newFieldPassword);
  boolean checkPassword (String customerId, String password);
  String getGrettings (String customerId);
  boolean createCustomer (String customerId, String password, String firstName, String lastName, String title);
  boolean removeCustomer (String customerId);
  boolean updateCustomer (String customerId, String password, String firstName, String lastName, String title);
  boolean findElement (String customerId);
} // interface ORBCustomer
```

ORBTrans.java:

```
package IDLDevelopment;

/**
* IDLDevelopment/ORBTrans.java
* Generated by the IBM IDL-to-Java compiler, version 1.0
* from trans.idl  Wednesday, July 28, 1999 6:41:18 o'clock PM EST
*/

public interface ORBTrans
{
  String fieldAcctNumber ();
  void fieldAcctNumber (String newFieldAcctNumber);
  int fieldNumber ();
  void fieldNumber (int newFieldNumber);
  String fieldType ();
  void fieldType (String newFieldType);
  String fieldDate ();
  void fieldDate (String newFieldDate);
  float fieldAmount ();
  void fieldAmount (float newFieldAmount);
  int transCount ();
  void transCount (int newTransCount);
  void getTransInfo (String customerId);
  boolean createTrans (String acctnumber, int number, String type, String date, float amount);
  boolean removeTrans (String acctnumber, int number);
  boolean findElement (String acctnumber, int number);
  void setTransNumberArray (String s, int i);
  String getTransNumberArray (int i);
  void setTransTypeArray (String s, int i);
  String getTransTypeArray (int i);
  void setTransDateArray (String s, int i);
  String getTransDateArray (int i);
  void setAmountArray (String s, int i);
  String getAmountArray (int i);
} // interface ORBTrans
```

SimpleCustomer.java:

```
package IDLDevelopment;

import java.sql.*;

public class SimpleCustomer implements IDLDevelopment.ORBCustomer {
        static String url = null;
        static String DbUserid="db2admin";
        static String DbPasswd="admin";
        static String JDBCDriver = "COM.ibm.db2.jdbc.app.DB2Driver";
        static Connection con;
        static
        {
                try
```

```
                {
                        Class.forName(JDBCDriver);
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        private String fieldCustomerId = new String();
        private String fieldFirstName = new String();
        private String fieldLastName = new String();
        private String fieldPassword = new String();
        private String fieldTitle = new String();


public SimpleCustomer() {
        super();
}

public String fieldCustomerId () {
        return fieldCustomerId;
}

public void fieldCustomerId (String newFieldCustomerId) {
        fieldCustomerId = newFieldCustomerId;
}

public String fieldFirstName () {
        return fieldFirstName;
}

public void fieldFirstName (String newFieldFirstName) {
        fieldFirstName = newFieldFirstName;
}

public String fieldLastName () {
        return fieldLastName;
}

public void fieldLastName (String newFieldLastName) {
        fieldLastName = newFieldLastName;
}

public String fieldTitle () {
        return fieldTitle;
}

public void fieldTitle (String newFieldTitle) {
        fieldTitle = newFieldTitle;
}

public String fieldPassword () {
        return fieldPassword;
}

public void fieldPassword (String newFieldPassword) {
        fieldPassword = newFieldPassword;
}

public boolean checkPassword (String acustomerId, String apassword) {
        boolean rc = false;
        String password = new String();
        Statement stmt= null;
        ResultSet rs =null;
        url="jdbc:db2:AIS";

        try {
                con=DriverManager.getConnection(url,DbUserid,DbPasswd);
        } catch (Exception e){
                e.printStackTrace();
        }
        try {
                stmt= con.createStatement();
                String query= "select cust_pwd from customer where cust_id = '"+acustomerId+"'";
                rs = stmt.executeQuery(query);
                if (rs.next()) {
```

```
                    password = rs.getString(1);
                    password = password.trim();
                }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        finally{
                try {
                    rs.close();
                    stmt.close();
                } catch (Exception e) {}
        }

        if (password.equals(apassword)) {
                rc = true;
        }
        return rc;
}

public String getGrettings (String acustomerId) {
        String grettings = new String();
        Statement stmt= null;
        ResultSet rs =null;
        url="jdbc:db2:AIS";

        try {
                con=DriverManager.getConnection(url,DbUserid,DbPasswd);
        } catch (Exception e){
                e.printStackTrace();
        }
        try {
                stmt= con.createStatement();
                String query= "select cust_fn, cust_ln from customer where cust_id = '"+acustomerId+"'";
                rs = stmt.executeQuery(query);
                if (rs.next()) {
                    grettings = rs.getString(1)+" "+rs.getString(2);
                }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally{
                try {
                    rs.close();
                    stmt.close();
                } catch (Exception e) {}
        }

        return grettings;
}

public boolean createCustomer(String acustomerId, String apassword, String afirstName, String alastName, String
atitle) {
        boolean rc = false;
        Statement stmt= null;
        url="jdbc:db2:AIS";

        try {
                con=DriverManager.getConnection(url,DbUserid,DbPasswd);
        } catch (Exception e){
                e.printStackTrace();
        }
        try {
                stmt= con.createStatement();
                String cmd = "insert into customer values('"+ acustomerId + "','" + alastName +"','" + afirstName
                +"','";
                cmd = cmd + atitle + "','" + apassword+"')";
                if (stmt.executeUpdate(cmd) != 0) {
                    rc = true;
                }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        finally{
                try {
                    stmt.close();
                } catch (Exception e) {}
```

```
		}

		return rc;
}

public boolean removeCustomer(String acustomerId) {
		boolean rc = false;
		Statement stmt= null;
		url="jdbc:db2:AIS";

		try {
			con=DriverManager.getConnection(url,DbUserid,DbPasswd);
		} catch (Exception e){
			e.printStackTrace();
		}
		try {
			stmt= con.createStatement();
			String cmd = "delete from customer where cust_id = '"+ acustomerId + "'";
			if (stmt.executeUpdate(cmd) != 0) {
				rc = true;
			}
		} catch (SQLException e) {
		    e.printStackTrace();
		}
		finally{
			try {
				stmt.close();
			} catch (Exception e) {}
		}

		return rc;

}
public boolean updateCustomer(String acustomerId, String apassword, String afirstName, String alastName, String
atitle) {
		boolean rc = false;
		Statement stmt= null;
		url="jdbc:db2:AIS";

		try {
			con=DriverManager.getConnection(url,DbUserid,DbPasswd);
		} catch (Exception e){
			e.printStackTrace();
		}
		try {
			stmt= con.createStatement();
			String cmd = "update customer set cust_fn = '" + afirstName + "',cust_ln = '" + alastName + "',";
			cmd = cmd + "cust_title = '" + atitle + "', cust_pwd = '" + apassword + "' ";
			cmd = cmd + "where cust_id = '"+ acustomerId + "'";
			if (stmt.executeUpdate(cmd) != 0) {
				rc = true;
			}
		} catch (SQLException e) {
		    e.printStackTrace();
		}
		finally{
			try {
				stmt.close();
			} catch (Exception e) {}
		}

		return rc;

}

public boolean findElement (String acustomerId) {
		boolean rc = false;
		Statement stmt= null;
		ResultSet rs =null;
		url="jdbc:db2:AIS";

		try {
			con=DriverManager.getConnection(url,DbUserid,DbPasswd);
		} catch (Exception e){
			e.printStackTrace();
```

```
		}
		try {
			stmt= con.createStatement();
			String query= "select * from customer where cust_id = '"+acustomerId+"'";
			rs = stmt.executeQuery(query);
			if (rs.next()) {
				fieldCustomerId(rs.getString(1));
				fieldLastName(rs.getString(2));
				fieldFirstName(rs.getString(3));
				fieldTitle(rs.getString(4));
				fieldPassword(rs.getString(6));
				rc = true;
			}
		} catch (SQLException e) {
		      e.printStackTrace();
		}
		finally{
			try {
				rs.close();
				stmt.close();
			} catch (Exception e) {}
		}

		return rc;
  }
}
```

SimpleTrans.java:

```
package IDLDevelopment;

import java.sql.*;
import java.util.*;
import java.text.*;

public class SimpleTrans implements IDLDevelopment.ORBTrans {
		private String acctnum = "0";
		static String url = null;
		static String DbUserid="db2admin";
		static String DbPasswd="admin";
		static String JDBCDriver = "COM.ibm.db2.jdbc.app.DB2Driver";
		static Connection con;
		static
		{
			try
			{
				Class.forName(JDBCDriver);
			} catch (Exception e) {
				e.printStackTrace();
			}
		}

		private String fieldAcctNumber = new String();
		private int fieldNumber = 0;
		private String fieldType = new String();
		private String fieldDate = new String();
		private float fieldAmount = 0;

		private int transCount = 0;
		private int numberOfTrans = 20;

		private String[] transNumberArray = new String[numberOfTrans];
		private String[] transTypeArray = new String[numberOfTrans];
		private String[] transDateArray = new String[numberOfTrans];
		private String[] amountArray = new String[numberOfTrans];

		private NumberFormat dollars = NumberFormat.getCurrencyInstance();

public SimpleTrans() {
		super();
}

public String fieldAcctNumber() {
```

```
        /* Perform the fieldNumber method. */
        return fieldAcctNumber;
}

public void fieldAcctNumber(String acctNumber) {
        /* Perform the fieldNumber method. */
        fieldAcctNumber = acctNumber;
        return;
}

public int fieldNumber() {
        /* Perform the fieldNumber method. */
        return fieldNumber;
}

public void fieldNumber(int number) {
        /* Perform the fieldNumber method. */
        fieldNumber = number;
        return;
}

public String fieldType() {
        /* Perform the fieldType method. */
        return fieldType;
}

public void fieldType(String type) {
        /* Perform the fieldType method. */
        fieldType = type;
        return;
}
public String fieldDate() {
        /* Perform the fieldDate method. */
        return fieldDate;
}

public void fieldDate(String date) {
        /* Perform the fieldDate method. */
        fieldDate = date;
        return;
}

public float fieldAmount() {
        /* Perform the fieldAmount method. */
        return fieldAmount;
}

public void fieldAmount(float amount) {
        /* Perform the fieldBalance method. */
        fieldAmount = amount;
        return;
}

public void setTransNumberArray(String s, int i) {
        transNumberArray[i] = s;
}

public String getTransNumberArray(int i) {
        return transNumberArray[i];
}

public void setTransTypeArray(String s, int i) {
        transTypeArray[i] = s;
}

public String getTransTypeArray(int i) {
        return transTypeArray[i];
}

public void setTransDateArray(String s, int i) {
        transDateArray[i] = s;
}

public String getTransDateArray(int i) {
        return transDateArray[i];
```

```
}

public void setAmountArray(String s, int i) {
	amountArray[i] = s;
}

public String getAmountArray(int i) {
	return amountArray[i];
}

public void transCount(int i) {
	transCount = i;
}

public int transCount() {
	return transCount;
}

public void getTransInfo(String acctNumber) {
	Statement stmt =null;
	ResultSet rs =null;
	url="jdbc:db2:AIS";

	try {
		con=DriverManager.getConnection(url,DbUserid,DbPasswd);
	} catch (Exception e) {
		e.printStackTrace();
	}
	try {
		stmt= con.createStatement();
		String query= "select txn_number, txn_type, txn_date, amount from db2admin.txn where
acct_number = '"+acctNumber+"' order by txn_number DESC";
		rs = stmt.executeQuery(query);
		transCount = 0;
		while (rs.next() && transCount < 20)  {
			setTransNumberArray(rs.getString(1), transCount);
			setTransTypeArray(rs.getString(2), transCount);
			setTransDateArray(rs.getString(3), transCount);
			setAmountArray(dollars.format(rs.getDouble(4)), transCount);
			transCount++;
		}
	} catch (SQLException e) {
		e.printStackTrace();
	}
	finally{
		try {
			rs.close();
			stmt.close();
		} catch (Exception e) {}
	}

}

public boolean createTrans(String acctnumber, int anumber, String atype, String adate, float aamount) {
	boolean rc = false;
	Statement stmt= null;
	url="jdbc:db2:AIS";

	try {
		con=DriverManager.getConnection(url,DbUserid,DbPasswd);
	} catch (Exception e){
		e.printStackTrace();
	}
	try {
		stmt= con.createStatement();
		String cmd = "insert into txn values('"+ acctnumber + "'," + anumber +"," + aamount +",'" + atype
		+"'" + adate +"')";
		if (stmt.executeUpdate(cmd) != 0) {
			rc = true;
		}
	} catch (SQLException e) {
	      e.printStackTrace();
	}
	finally{
		try {
```

```
                        stmt.close();
                } catch (Exception e) {}
        }

        return rc;
}

public boolean removeTrans(String acctnumber, int anumber) {
        boolean rc = false;
        Statement stmt= null;
        url="jdbc:db2:AIS";

        try {
                con=DriverManager.getConnection(url,DbUserid,DbPasswd);
        } catch (Exception e){
                e.printStackTrace();
        }
        try {
                stmt= con.createStatement();
                String cmd = "delete from txn where acct_number = '"+ acctnumber + "' and txn_number =
                "+anumber;
                if (stmt.executeUpdate(cmd) != 0) {
                        rc = true;
                }
        } catch (SQLException e) {
              e.printStackTrace();
        }
        finally{
                try {
                        stmt.close();
                } catch (Exception e) {}
        }

        return rc;

}

public boolean findElement (String acctnumber, int anumber) {
        boolean rc = false;
        Statement stmt= null;
        ResultSet rs =null;
        url="jdbc:db2:AIS";

        try {
                con=DriverManager.getConnection(url,DbUserid,DbPasswd);
        } catch (Exception e){
                e.printStackTrace();
        }
        try {
                stmt= con.createStatement();
                String query= "select * from db2admin.txn where acct_number = '"+ acctnumber + "' and
                txn_number = "+anumber;
                rs = stmt.executeQuery(query);
                if (rs.next()) {
                        fieldAcctNumber(rs.getString(1));
                        fieldNumber(rs.getInt(2));
                        fieldType(rs.getString(4));
                        fieldDate(rs.getString(5));
                        fieldAmount(rs.getFloat(3));
                        rc = true;
                }
        } catch (SQLException e) {
              e.printStackTrace();
        }
        finally{
                try {
                        rs.close();
                        stmt.close();
                } catch (Exception e) {}
        }

        return rc;
 }
}
```

SimpleAccount.java:

```
package IDLDevelopment;

import java.sql.*;
import java.util.*;
import java.text.*;

public class SimpleAccount implements IDLDevelopment.ORBAccount {
//public class SimpleAccount {
	private String acctnum = "0";
	static String url = null;
	static String DbUserid="db2admin";
	static String DbPasswd="admin";
	static String JDBCDriver = "COM.ibm.db2.jdbc.app.DB2Driver";
	static Connection con;
	static
	{
		try
		{
			Class.forName(JDBCDriver);
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	private String fieldNumber = new String();
	private String fieldType = new String();
	private String fieldCustomerId = new String();
	private float fieldBalance = 0;

	private int accountCount = 0;
	private int numberOfAccounts = 4;

	private String[] accountNumberArray = new String[numberOfAccounts];
	private String[] accountTypeArray = new String[numberOfAccounts];
	private String[] balanceArray = new String[numberOfAccounts];

	private NumberFormat dollars = NumberFormat.getCurrencyInstance();

public SimpleAccount() {
	super();
}

public float deposit(float howMuch, String acctnum) {
	/* Perform the deposit method. */
	float aBalance = getBalance(acctnum);
	setBalance(aBalance+howMuch, acctnum);
	saveTrans(acctnum, "DP", howMuch);

	return getBalance(acctnum);
}

public float transfer(float howMuch, String acctnum1, String acctnum2) {
	if (!acctnum2.equals(acctnum1)) {
		/* Perform the withdraw method. */
		float aBalance1=getBalance(acctnum1);
		if (howMuch < aBalance1) {
			setBalance(aBalance1-howMuch,acctnum1);
			saveTrans(acctnum1, "TW", howMuch);

			/* Perform the deposit method. */
			float aBalance2 = getBalance(acctnum2);
			setBalance(aBalance2+howMuch, acctnum2);
			saveTrans(acctnum2, "TD", howMuch);
		}
	}
	return getBalance(acctnum2);
}

public String fieldNumber() {
	/* Perform the fieldNumber method. */
	return fieldNumber;
}

public void fieldNumber(String number) {
```

```
		/* Perform the fieldNumber method. */
		fieldNumber = number;
		return;
}

public String fieldCustomerId() {
		/* Perform the fieldCustomerId method. */
		return fieldCustomerId;
}

public void fieldCustomerId(String customerId) {
		/* Perform the fieldNumber method. */
		fieldCustomerId = customerId;
		return;
}
public String fieldType() {
		/* Perform the fieldType method. */
		return fieldType;
}

public void fieldType(String type) {
		/* Perform the fieldType method. */
		fieldType = type;
		return;
}

public float fieldBalance() {
		/* Perform the fieldBalance method. */
		return fieldBalance;
}

public void fieldBalance(float balance) {
		/* Perform the fieldBalance method. */
		fieldBalance = balance;
		return;
}

public float getBalance(String acctnum) {
		String balanceS;
		Float balanceF;
		Statement stmt= null;
		ResultSet rs =null;
		url="jdbc:db2:AIS";

		try {
			con=DriverManager.getConnection(url,DbUserid,DbPasswd);
		} catch (Exception e){
			e.printStackTrace();
		}
		try {
			stmt= con.createStatement();
			String query= "select balance from db2admin.account where acct_number =
			'"+acctnum+"'";
			rs = stmt.executeQuery(query);
			if (rs.next()) {
				balanceS = rs.getString(1);
				balanceF = new Float(balanceS);
				fieldBalance(balanceF.floatValue());
			}
		} catch (SQLException e) {
		      e.printStackTrace();
		}
		finally{
			try {
				rs.close();
				stmt.close();
			} catch (Exception e) {}
		}

		return fieldBalance();
}

public void setBalance(float balance, String acctnum) {
		Statement stmt =null;
		url="jdbc:db2:AIS";
```

```
		try {
			con=DriverManager.getConnection(url,DbUserid,DbPasswd);
		} catch (Exception e) {
			e.printStackTrace();
		}
		try{
			stmt= con.createStatement();
			String query= "update account set balance = "+ balance +" where acct_number = '"
			+acctnum+"'";
			int updcnt = stmt.executeUpdate(query);
		} catch (SQLException e){
			e.printStackTrace();
		}
		finally{
			try{
				stmt.close();
			} catch (Exception e) {}
		}

		fieldBalance(balance);
}

public float withdraw(float howMuch,String acctnum) {
		/* Perform the withdraw method. */
		float aBalance=getBalance(acctnum);
		if (howMuch < aBalance) {
			setBalance(aBalance-howMuch,acctnum);
			saveTrans(acctnum, "WD", howMuch);
		}
		return getBalance(acctnum);
}

public void setAccountNumberArray(String s, int i) {
		accountNumberArray[i] = s;
}

public String getAccountNumberArray(int i) {
		return accountNumberArray[i];
}

public void setAccountTypeArray(String s, int i) {
		accountTypeArray[i] = s;
}

public String getAccountTypeArray(int i) {
		return accountTypeArray[i];
}

public void setBalanceArray(String s, int i) {
		balanceArray[i] = s;
}

public String getBalanceArray(int i) {
		return balanceArray[i];
}

public void accountCount(int i) {
		accountCount = i;
}

public int accountCount() {
		return accountCount;
}

public void getAccountInfo(String acustomerId) {
		Statement stmt =null;
		ResultSet rs =null;
		url="jdbc:db2:AIS";

		try {
			con=DriverManager.getConnection(url,DbUserid,DbPasswd);
		} catch (Exception e) {
			e.printStackTrace();
		}
		try {
```

```
                stmt= con.createStatement();
                String query= "select acct_number, acct_type, balance from db2admin.account where
                cust_id = '"+acustomerId+"'";
                rs = stmt.executeQuery(query);
                accountCount = 0;
                while (rs.next())  {
                        setAccountNumberArray(rs.getString(1), accountCount);
                        setAccountTypeArray(rs.getString(2), accountCount);
                        setBalanceArray(dollars.format(rs.getDouble(3)), accountCount);
                        accountCount++;
                }
        } catch (SQLException e) {
                e.printStackTrace();
        }
        finally{
                try {
                        rs.close();
                        stmt.close();
                } catch (Exception e) {}
        }

}

public boolean createAccount(String anumber, String acustomerId, String atype, float abalance) {
        boolean rc = false;
        Statement stmt= null;
        ResultSet rs =null;
        url="jdbc:db2:AIS";
        int count = 0;
        String query = new String();

        try {
                con=DriverManager.getConnection(url,DbUserid,DbPasswd);
        } catch (Exception e){
                e.printStackTrace();
        }
        try {
                stmt= con.createStatement();
                if (anumber.equals("") || anumber == null) {
                        query= "select max(int(acct_number))+1 from account";
                        System.out.println("query:"+query);
                        rs = stmt.executeQuery(query);
                        if (rs.next()) {
                                anumber = rs.getString(1).trim();
                                System.out.println("anumber:"+anumber);
                        }

                }

                query= "select count(*) from account where cust_id = '" + acustomerId + "'";
                System.out.println("query:"+query);
                rs = stmt.executeQuery(query);
                if (rs.next()) {
                        count = rs.getInt(1);
                        System.out.println("count:"+count);
                }

                if (count < 4 && abalance > 0) {
                        String cmd = "insert into account values('"+ anumber + "','" + acustomerId
                                +"','" + atype +"'," + abalance +")";
                        if (stmt.executeUpdate(cmd) != 0) {
                                saveTrans(anumber, "DP", abalance);
                                rc = true;
                        }
                }

        } catch (SQLException e) {
              e.printStackTrace();
        }
        finally{
                try {
                        stmt.close();
                } catch (Exception e) {}
        }
```

```
		return rc;
	}

	public boolean removeAccount(String anumber) {
		boolean rc = false;
		Statement stmt= null;
		url="jdbc:db2:AIS";

		try {
			con=DriverManager.getConnection(url,DbUserid,DbPasswd);
		} catch (Exception e){
			e.printStackTrace();
		}
		try {
			stmt= con.createStatement();
			String cmd = "delete from account where acct_number = '"+ anumber + "'";
			if (stmt.executeUpdate(cmd) != 0) {
				rc = true;
			}
		} catch (SQLException e) {
		      e.printStackTrace();
		}
		finally{
			try {
				stmt.close();
			} catch (Exception e) {}
		}

		return rc;
	}

	public boolean findElement (String anumber) {
		boolean rc = false;
		Statement stmt= null;
		ResultSet rs =null;
		url="jdbc:db2:AIS";

		try {
			con=DriverManager.getConnection(url,DbUserid,DbPasswd);
		} catch (Exception e){
			e.printStackTrace();
		}
		try {
			stmt= con.createStatement();
			String query= "select * from db2admin.account where acct_number = '"+anumber+"'";
			rs = stmt.executeQuery(query);
			if (rs.next()) {
				fieldNumber(rs.getString(1));
				fieldCustomerId(rs.getString(2));
				fieldType(rs.getString(3));
				fieldBalance(rs.getFloat(4));
				rc = true;
			}
		} catch (SQLException e) {
		      e.printStackTrace();
		}
		finally{
			try {
				rs.close();
				stmt.close();
			} catch (Exception e) {}
		}

		return rc;
	}

	public boolean saveTrans(String acctnumber, String atype, float aamount) {
		boolean rc = false;
		Statement stmt= null;
		ResultSet rs =null;
		url="jdbc:db2:AIS";
		//java.util.Date today = new java.util.Date();
		//java.sql.Date adate = new java.sql.Date(today.getYear(),today.getMonth(),today.getDate());
```

```
        java.sql.Date adate = new java.sql.Date(Calendar.getInstance().get(Calendar.YEAR)-
1900,Calendar.getInstance().get(Calendar.MONTH),Calendar.getInstance().get(Calendar.DAY_OF_MONT
H));
        String anumber = new String();
        if (aamount > 0) {
                try {
                        con=DriverManager.getConnection(url,DbUserid,DbPasswd);
                } catch (Exception e){
                        e.printStackTrace();
                }

                try {
                        stmt= con.createStatement();
                        String query= "select max(txn_number)+1 from txn where acct_number =
                                '"+acctnumber + "'";
                        System.out.println("query:"+query);
                        rs = stmt.executeQuery(query);
                        if (rs.next()) {
                                anumber = rs.getString(1);
                        }
                        if (anumber == null) {
                                anumber = "1";
                        }
                        System.out.println("anumber:"+anumber);

                        String cmd = "insert into txn values('"+ acctnumber + "'," + anumber +"," +
                                aamount +",'" + atype +"','" + adate.toString() + "')";
                        System.out.println("cmd:"+cmd);
                        if (stmt.executeUpdate(cmd) != 0) {
                                rc = true;
                        }
                } catch (SQLException e) {
                e.printStackTrace();
                }
                finally{
                        try {
                                rs.close();
                                stmt.close();
                        } catch (Exception e) {}
                }
        }

        return rc;
  }
}
```

BIBLIOGRAFÍA.

1. AKERLEY John, HASHIM Murtuza, Developing an e-Business Application for the Websphere Application Server, IBM International Technical Support Organization, SG24-5423-00, Julio 1999.

2. CAMPIONE Mary y WALRATH Kathy, The Java Tutorial Object Oriented Programming for the Internet, Addison Wesley, 2a. Edición.

3. GRAF Olaf, KOTSEN Avril, Visual Age for Java Enterprise Version 2: Data Access Beans – Servlets – CICS Connector, IBM International Technical Support Organization, SG24-5265-00, Diciembre 1998.

4. ORFALI, Robert y HARKEY Robert, Client / Server Programming with Java and Corba, Wiley Computer Publishing, 2a. Edición.

5. ORFALI, Robert y HARKEY Robert, The Essential Client / Server Survival Guide, Wiley Computer Publishing, 2a. Edición.

6. PICON Joaquin y EDUARDS Craig, Using Visual Age for Java Enterprise Version 2 to Develop Corba and EJB Applications, IBM International Technical Support Organization, SG24-5276-00, Diciembre 1998.