ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"DESARROLLO DE UNA APLICACIÓN CLIENTE/SERVIDOR BASADA EN CORBA SISTEMA DE RESERVACIONES DE VUELO PARA UNA AEROLÍNEA"

EXAMEN DE GRADO (COMPLEXIVO)

Previa a la obtención del GRADO de:

INGENIERO EN COMPUTACIÓN

ERIC YOBANNY GUAGUA ALVARADO

GUAYAQUIL – ECUADOR

AÑO: 2015

AGRADECIMIENTO

A Dios primero y a las muchas personas especiales a las que me gustaría agradecer; su amistad, apoyo, ánimo y compañía en las diferentes etapas de mi vida. Algunas están aquí conmigo y otras en mis recuerdos y en el corazón. Sin importar en donde estén quiero darles las gracias por formar parte de mí, por todo lo que me han brindado y por todas sus bendiciones.

ERIC YOBANNY GUAGUA ALVARADO

DEDICATORIA

Dedico este trabajo primero a Dios por haberme permitido llegar hasta este momento con vida y haberme dado salud, ser el manantial de vida y darme lo necesario para seguir adelante día a día para lograr mis objetivos, además de su infinita bondad y amor, a mi madre María Alvarado por sus consejos, estímulo y su gran amor a mi padre Alcasí Guagua por impregnarme de su sabiduría y consejos , a mi esposa Silvia a mis hijos Santiago, Stefanic, Gabriela y Leonella que son la motivación constante que me ha permitido ser una persona de bien y luchar por este objetivo que al fin he culminado, pero más que nada, por el amor que he percibido de ellos.

ERIC YOBANNY GUAGUA ALVARADO

TRIBUNAL DE SUSTENTACIÓN

Ph.D. Sixto García	**M.Sc. Gonzalo Luzardo**
EVALUADOR	EVALUADOR

DECLARACIÓN EXPRESA

“La responsabilidad por los hechos, ideas y doctrinas expuestas en este Informe me corresponde exclusivamente; y, el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL”

(Reglamento de Graduación de la ESPOL).

Eric Yobanny Guagua Alvarado

RESUMEN

El proyecto realizado involucra el desarrollo de una aplicación de un sistema de reservaciones de vuelo para una aerolínea ficticia, en la que se aplicó una tecnología que permite hacer uso de objetos distribuidos. Se llama CORBA, la arquitectura común de corredores de solicitudes de objetos. Esta aplicación tiene una parte administrativa de tipo standalone (independiente) para uso interno de la compañía, y una parte para los clientes de la misma de tipo dependiente de browser (un applet de java) para brindar el servicio a través del web. El sistema permite hacer las siguientes transacciones: reserva, cancelación de reservas, ingreso y cancelación de vuelos; ingreso de recorridos, ingreso de ubicaciones y consultas.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

INTRODUCCIÓN

La computación ha evolucionado muchísimo en las dos últimas décadas, tanto a nivel de hardware como a nivel de software. En cuanto al software, las maneras de construir sistemas han cambiado desde que se introdujeron nuevas metodologías de diseño como la programación orientada a objetos y la arquitectura cliente/servidor. Esta última ha sido fundamental para el desarrollo realmente distribuidos haciendo que los programadores e ingenieros de software dejen a un lado viejos esquemas, como el de file server y más aún el de multiusuario y centren sus miradas en esta nueva forma de diseño. Por otro lado, la programación orientada a objetos ha suplantado otro tipo de programación conocida como programación estructurada encapsulando código y datos en un solo ente inteligente conocido como objeto.

Cualquier persona relacionada con las computadoras sabe que los objetos son maravillosos, y que sencillamente no podríamos vivir sin ellos. Hoy en día, existe un sin número de objetos mucha bibliografía relacionada a los conceptos como el encapsulado, la herencia y el polimorfismo. Nunca la programación orientada a objetos y la arquitectura cliente/servidor se complementaron tanto como en la actualidad.

En los primeros días los expertos en la industria de la computación se daban cuenta de las bondades que ofrecían los objetos y se preguntaban con insistencia ¿Qué pueden hacer los objetos en una red? ¿Cómo pueden apoyar estos al esquema cliente/servidor? Para decirlo brevemente ellos necesitan comprender como extender la tecnología de objetos para manejar los complejos asuntos inherentes a la creación

de robustos sistemas cliente/servidor. Ahora, cuando iniciamos un nuevo siglo, conocemos las respuestas a estas preguntas y sabemos con precisión qué pueden hacer los objetos por los sistemas con arquitectura cliente/servidor. Esta es la intención del presente proyecto, demostrar que en lo subsecuente la siguientes generaciones de sistemas cliente/ servidor estarán basados en objetos distribuidos.

Convencido de que está es el área en la que los objetos desarrollaran todo su potencial. También creo que internet, las intranets y las extranets necesitan de los objetos distribuidos. Obviamente pues, la computación en internet no sustituye de ninguna manera a la arquitectura cliente/ servidor, y esto se debe a que ya es en sí misma cliente/servidor.

Con el fin de cumplir con el propósito que mencionamos anteriormente, se ha desarrollado un “Sistema de Reservaciones de Vuelo” para una aerolínea ficticia donde hemos empleado una tecnología que permite hacer uso de objetos distribuidos. Se llama CORBA, la arquitectura común de corredores de solicitudes de objetos. Esta aplicación tiene una parte administrativa de tipo standalone para uso interno de la compañía, y una parte para los clientes de la misma de tipo dependiente de browser (un applet de java) para brindar el servicio a través del web. Con este sistema se busca facilitarle el trabajo al administrador en muchos campos del negocio y también el sistema proporciona algunos beneficios al cliente como el ahorro de tiempo.

CAPÍTULO 1

1. M E T O D O L O G Í A O SOLUCIÓN TECNOLÓGICA IMPLEMENTADA

1.1 Metodología

El desarrollo rápido de aplicaciones o RAD (acrónimo en inglés de rapid application development) es un proceso de desarrollo de software, desarrollado inicialmente por James Martin en1980. Permite construir sistemas utilizables en poco tiempo, el método comprende el desarrollo interactivo, la construcción de prototipos y el uso de *ingeniería de software asistida por computadora o sea* utilidades CASE (Computer Aided Software Engineering). ya definidas. . El Desarrollo Rápido de Aplicaciones es una adaptación a “Alta velocidad del modelo lineal secuencial” en el que se logra el desarrollo rápido utilizando un enfoque de construcción basado en

componentes. Si se comprenden bien los requisitos y se limita el ámbito del proyecto, el proceso RAD permite al equipo de desarrollo crear un “sistema completamente funcional” dentro de periodos cortos de tiempo. Tradicionalmente, el desarrollo rápido de aplicaciones tiende a englobar también la usabilidad, utilidad y la rapidez de ejecución. Mientras menos tiempo transcurre en el desarrollo del sistema menos habrán cambiado las necesidades de los usuarios.

Hoy en día se suele utilizar para referirnos al desarrollo rápido de interfaces gráficas de usuario tales como Glade, o entornos de desarrollo integrado completos. Algunas de las plataformas más conocidas son Visual Studio, Lazarus, Gambas, Game o Clarion. En el área de la autoría multimedia, software como Neosoft, Neoboo y MediaChance, Multimedia Builder proveen plataformas de desarrollo rápido de aplicaciones, dentro de ciertos límites.

La metodología de desarrollo de software diseño rápido de aplicaciones RAD consiste de diferentes etapas (Figura 1.1) que suceden de forma paralela y exigen la colaboración de los usuarios en todos los niveles, esta metodología propone un proceso de desarrollo de "software" que permite que se creen sistemas de computadoras utilizables en un periodo de tiempo entre 60 a 90 días. RAD es un ciclo de desarrollo diseñado para crear aplicaciones de computadoras de alta calidad de las que acontecen en grandes corporaciones.

El Modelo RAD comprende las siguientes **etapas**:

Modelado de gestión. Este modelo se basa en dar respuesta a las siguientes preguntas:

¿Qué información conduce el proceso de gestión?

¿Qué información genera?

¿A dónde va la información?

¿Quién la procesa?

Modelado de datos. En este modelo se definen los almacenes de datos y cómo se relacionan los almacenes entre sí.

Modelado del proceso. Se utiliza para añadir, modificar, suprimir o recuperar un objeto de datos.

Generación de aplicaciones. Para esto se utiliza una herramienta de cuarta generación que permite crear el software y facilitar la construcción del programa.

Pruebas y entrega. El proceso de desarrollo finaliza realizando pruebas de calidad del software diseñado con la herramienta RAD, posteriormente se realiza la implementación de la aplicación

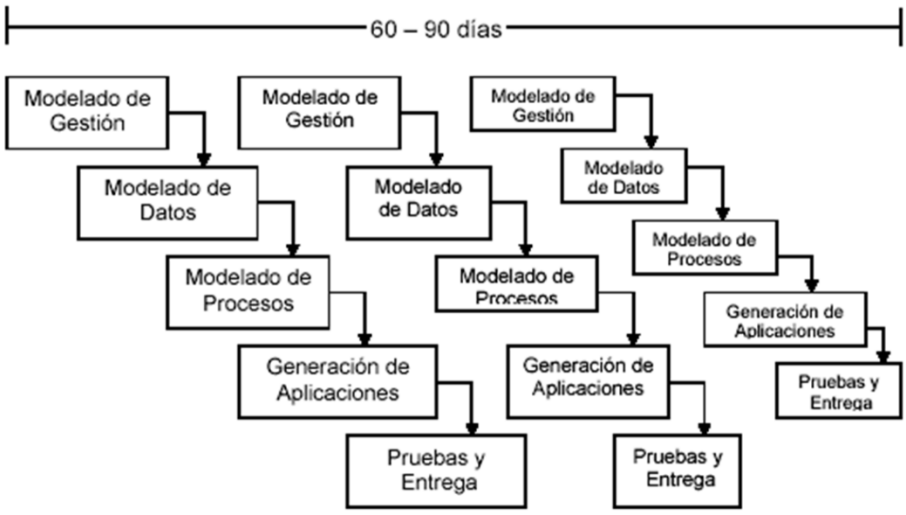

Figura 1.1. Fases del modelo desarrollo rápido de aplicaciones (RAD)

En este entorno de programación se desarrolló el presente proyecto.

1.2 Especificaciones

Las especificaciones se detallan a continuación:

Debe ser desarrollado con arquitectura cliente/servidor haciendo uso de un esquema de tres capas

Debe hacer uso de la tecnología de objetos distribuidos y demostrar su importancia para la nueva generación de sistemas con arquitectura cliente/servidor de aquel entonces.

Debe emplear la tecnología que nos ofrece la arquitectura CORBA y lo conveniente que es para el desarrollo de sistemas cliente/servidor de tres capas

Debe mostrar la aplicación que tiene un DBMS cliente/servidor en un ambiente tres capas

1.3 Requerimientos funcionales

Se desea crear una aplicación para que sea accesible desde el Web, para lo cual se utilizará applets de Java. Los applets de Java podrán ser ejecutados desde distintas maquinas clientes. El tema principal del proyecto es el desarrollo de un sistema cliente/servidor para una aerolínea ficticia usando tecnología de Internet. Las maquinas clientes deben contar con la existencia de browsers los cuales descargaran del servidor un applet de java en el cual se brindan las opciones de los distintos tipos de transacciones con los que contara el proyecto.

Las disposiciones para realizar este proyecto son las de hacer uso del lenguaje de programación Java y el uso de CORBA como middleware de objetos.

Del análisis del sistema se determina que el proyecto ha sido concebido para ejecutarse en una intranet o internet, por lo que el programa deberá utilizar applets y ejecutarse en un browser. Además este applet debe proveer la interfaz gráfica necesaria para brindarle al usuario la posibilidad de realizar las siguientes cinco transacciones que de lo analizado y de la información recogida, se identificaron las cuales son las metas “primordiales” del sistema.

1. Reserva
2. Cancelación de reservas

3. Ingreso y cancelación de vuelos
4. Ingreso de recorridos
5. Ingreso de ubicaciones y consultas

1.3.1 Requerimientos del proceso servidor

1. Utilizar el modelo de procesamiento paralelo Pool de objetos de servidor para implementar el proceso servidor y que éste pueda atender los requerimientos de los procesos clientes
2. Implementar parte de la lógica de la aplicación en procedimientos almacenados (stored procedures) usando un manejador de bases de datos (DBMS) Cliente/servidor.
3. Emplear CORBA usando los ORB (Object Request Broker) que es el bus de comunicación de objetos de Inprise Visibroquer como middleware en nuestro ambiente Cliente/Servidor

1.3.2 Requerimientos para el administrador

1. Crear las ciudades que serán tomadas en cuenta en los recorridos de los respectivos vuelos
2. Realizar la respectiva consulta de las ciudades
3. Crear los recorridos para los vuelos que seguirán los respectivos aviones ingresando la distancia en kilómetros entre los puntos origen y destino y el respectivo costo
4. Realizar la respectiva consulta de recorridos

5. Crear los vuelos escogiendo el recorrido apropiado y adicionalmente ingresando el número de asientos disponibles y la fecha y hora de partida y llegada del vuelo
6. Realizar la respectiva consulta de vuelos

1.3.3 Requerimiento para el cliente de la aerolínea

1. Deberá registrarse la primera vez que desee hacer una reservación, la siguiente ocasión que desee usar el sistema bastara con que valide su ingreso en la opción de autenticación
2. Podrá realizar las respectivas reservaciones escogiendo el vuelo que desee tomar y adicionalmente ingresando el número de asientos que desee reservar. El sistema devolverá el costo que debe pagar
3. Existe la opción de poder cancelar las reservaciones.

1.4 Restricciones del sistema

Para la implementación del sistema lo mínimo necesario en cuanto a **software** es:

- Utilizar cualquier sistema operativo Windows
- Un Browser Java – enabled
- Un compilador java jdk 1.1.7
- Microsoft SQL Server 7.0 para la Base de Datos
- Inprise Visibroker 3.1 para los ORB de CORBA

En cuanto al **hardware** lo mínimo necesario es:

Procesador

Pentium de 100 MHz (en el año 1999) como mínimo pero Pentium II 200 MHz recomendable para las computadoras donde van a correr tanto los procesos clientes como el proceso servidor

Memoria

24 MB (en el año 1999) donde corra el proceso servidor como mínimo, pero recomendable 32 Mb y 16Mb en aquellas computadoras donde corren los procesos clientes

El sistema esta implementado en java que es un lenguaje que permite hacer aplicaciones para la web así como aplicaciones de red en general, lo que significa que es idóneo para el desarrollo de sistemas distribuidos con arquitectura cliente/servidor.

La base de datos fue implementada en el producto SQL Server 7.0 de Microsoft, que posee un poderoso motor de base de datos que inclusive tiene soporte de terabytes y un sinnúmero de características que facilitan su administración, además de un fabuloso esquema de seguridad.

1.5 Esquema de la Arquitectura de la Aplicación

El Sistema de Reservaciones de Vuelo posee una arquitectura Cliente/Servidor de Tres Capas. En la que se debía aplicar la tecnología que

permite hacer uso de objetos distribuidos llamada arquitectura común de corredores de solicitudes de objetos CORBA (por sus siglas en inglés Common Object Request Broker Arquitecture)

Las tres capas cconsiste en una capa de la Presentación, otra capa de la lógica de la aplicación y otra capa de la base de datos. Normalmente esta arquitectura se utiliza en las siguientes situaciones:

- Cuando se requiera mucho procesamiento de datos en la aplicación.
- En aplicaciones donde la funcionalidad este en constante cambio.
- Cuando los procesos no están relativamente muy relacionados con los datos.
- Cuando se requiera aislar la tecnología de la base de datos para que sea fácil de cambiar.
- Cuando se requiera separar el código del cliente para que se facilite el mantenimiento.
- Es muy adecuada para utilizarla con la tecnología orientada a objetos.

El desarrollo de este tipo de aplicaciones requiere de conocimientos en una serie de áreas como el procesamiento de transacciones, diseño de base de datos, protocolos de comunicaciones y conocimiento de la interfaz gráfica del usuario. Las aplicaciones más avanzadas requieren conocimientos de objetos distribuidos e Internet. Debemos entonces mencionar algo de estos conceptos.

1.5.1 ¿Qué es cliente/servidor?

Clientes y servidores son entidades lógicas independientes que operan en conjunto ya sea a través de una red o en la misma computadora para realizar una tarea específica. A continuación presentamos una definición más formal:

Cliente/Servidor es una arquitectura de diseño de software que subdivide la aplicación en un conjunto de procesos servidores, generalmente especializados, que pueden ejecutarse en variadas plataformas (hardware+software), que proveen servicios (datos, información, procesamiento, etc.) a un conjunto de procesos clientes, que pueden ejecutarse en diferentes plataformas (hardware+software), a través de redes de área local o de redes de área extendida, utilizando uno o varios protocolos de comunicación.

1.5.2 ¿Qué es CORBA?

CORBA (Common Object Request Broker Arquitecture) es un estándar para objetos distribuidos que ha sido desarrollado por el OMG (Object Management Group), una asociación de empresas y usuarios de las más grandes que existen actualmente, que agrupa a unas 800 entidades que se dedican básicamente al desarrollo y comercialización de software para sistemas informáticos, y, que en su mayoría, se encuentran actualmente desarrollando aplicaciones que soportan este estándar o comercializando productos que ya lo hacen.

CORBA proporciona los mecanismos a través de los cuales los objetos hacen peticiones y reciben respuestas, definidas como ORBs (Object Request Broker) de forma transparente para el sistema. El ORB de CORBA es entonces, una aplicación que proporciona interoperabilidad entre diferentes objetos posiblemente programados en diferentes lenguajes, corriendo bajo sistemas operativos distintos y en general en diferentes máquinas. Este es el punto básico en el que se basa la arquitectura CORBA (Figura 1.2).

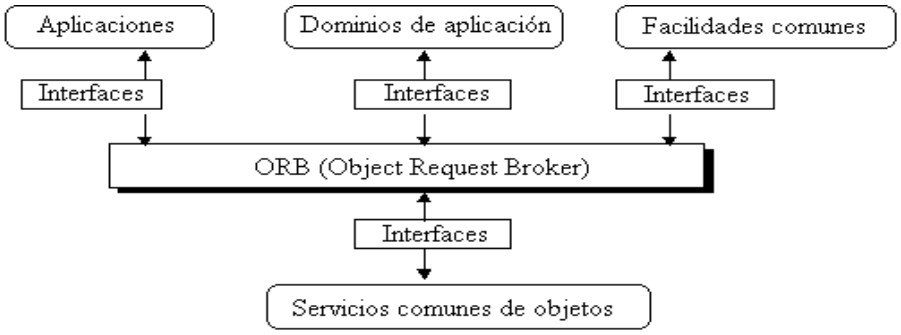

Figura 1.2 Arquitectura de CORBA

1.5.2.1 ¿Qué es ORB?

El ORB (Object Request Broker), intermediador de requerimientos de servicios, es el middleware que permite la comunicación entre cliente y servidor por medio de objetos. Usando un ORB, un cliente puede transparentemente invocar un método en un objeto servidor, el cual puede estar en una misma máquina o en algún lugar de la red. El ORB intercepta la llamada y es el responsable de encontrar un objeto que pueda implementar el requerimiento, pasar los parámetros, invocar el

método, y retornar los resultados. El cliente no necesita conocer donde está localizado el objeto, su lenguaje de programación, sistema operativo, o algún otro aspecto del sistema que no sea parte de la interface del objeto. El ORB es simplemente un proceso cuya función es localizar los procesos que proveerán los servicios requeridos. Una vez localizados el cliente puede conectarse con el servidor. Los objetos en el ORB pueden actuar tanto como cliente o servidor, dependiendo de la ocasión.

La comunicación entre los ORBs se lo realiza por medio del protocolo IIOP (Internet Inter-ORB Protocol).

1.5.2.2 IIOP (Internet Inter-ORB Protocol)

OMG ha definido un conjunto de reglas que dan formato a los datos que se transfieren, llamado CDR (Common Data Representation), además de un conjunto de tipos de mensajes. Juntos, los CDR y los tipos de mensajes, constituyen un protocolo abstracto llamado GIOP (General Inter-ORB Protocol). Para lograr verdadera interoperabilidad entre objetos distribuidos que residen en ambientes heterogéneos sobre Internet, se necesita que **ORBS** envíen mensajes a través de TCP/IP, porque es el protocolo de transporte orientado a conexión estándar para Internet. En pocas palabras, al unir GIOP + TCP/IP da como resultado el protocolo IIOP.

Para otros ambientes que no son TCP/IP, se utiliza el protocolo ESIOP (Environment Specific Inter-ORB Protocol).

Los objetos publican sus identidades y ubicaciones utilizando las referencias de objetos. CORBA especifica un formato común para las

referencias de objetos llamado IOR (Interoperable Object Reference), el cual contiene perfiles que describen como los clientes pueden encontrar y enviar requerimientos a1 servidor usando un protocolo particular.

Todos los IORs, deben tener por lo menos un perfil IIOP, lo cual asegura que donde sea que se encuentre la referencia de objeto, cualquier ORB que cumpla con CORBA, será capaz de localizar el servidor y enviarle los requerimientos. El perfil IIOP tiene la dirección Internet del servidor y un valor clave usado por el servidor para encontrar el objeto especifico descrito por la referencia.

IIOP brinda un ambiente en el cual 10s programadores definen e invocan cualquier método que necesitan, y transparentemente se realiza la comunicación, es decir, los programadores no tienen que escribir programas IIOP, nunca requieren interactuar con IIOP de ninguna forma, simplemente es invisible para ellos.

1.5.2.3 Estructura de un ORB de CORBA

a) En el Cliente

Stubs del lado del cliente: Los stubs precompilados definen como los clientes invocan a los correspondientes servicios en los servidores (véase la figura 1.3). Del lado del cliente el stub actúa como una llamada local. Los servicios son definidos usando IDL(lenguaje de definición de interfaces), y ambos stubs tanto del cliente como del servidor son generados por el compilador IDL.

Dynamic Invocation Interface (DII): Permite descubrir métodos para ser invocados en tiempo de corrida.

Interface Repository: el depósito de interfaces es una base de datos que almacena información de las interfaces definidas. Permite obtener y modificar las descripciones de todas las interfaces de componentes registrados, los métodos que soportan y los parámetros que ellos requieren.

ORB interface: Consiste de unos pequeños APIs para servicios locales que pueden ser útiles en una aplicación. Por ejemplo APIs para convertir una referencia de objeto a string y viceversa.

b) En el Servidor

El Server IDL Stubs: (OMG los llama skeletons) proporcionan interfaces estáticas para cada servicio exportado por el servidor. Son creados usando el compilador IDL.

Implementation Repository: almacena información que permite al ORB localizar implementación de objetos.

Dynamic Skeleton Interface (DSI): Fueron introducidos por Corba 2.0, proporcionan un mecanismo de enlace en tiempo de ejecución para servidores que necesitan manejar invocación de métodos de entrada para componentes que no tienen skeletons compilados basados en IDL (o stubs). Los Dynamic Skeletons son muy usados para implementar los enlaces entre ORBs.

Object Adapter: es el medio que permite la comunicación entre el ORBCore y el OI (object implementation).

ORB Interface: Consiste de unos pocos APIs para servicios locales que son idénticos a los proporcionados en el cliente.

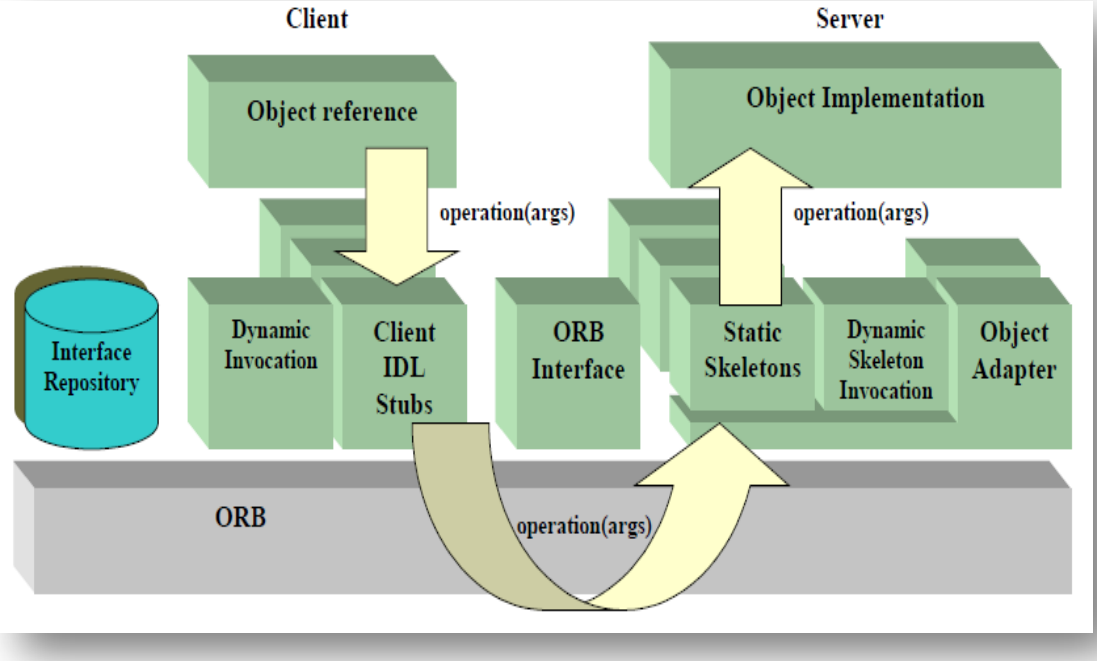

Figura 1.3 Invocación general de CORBA

1.5.2.4 Tipos de invocaciones del ORB

El Object Request Broker (ORB) realiza dos tipos de Invocaciones:

- Invocación Estática
- Invocación Dinámica

Invocación Estática

Se la define como una interface, la cual contiene descripción de: atributos (variables), métodos que van a ser invocados por el cliente y los argumentos de los métodos.

La invocación estática es implementada en IDL (Interface Definition Language), lenguaje que es puramente declarativo donde se define el nombre del método junto con sus argumentos que van ser invocados por el cliente (figura 1.4).

Invocación Dinámica.

Antes de llamar a un objeto dinámicamente debemos primeramente obtener el objeto y su referencia, dicha referencia nos servirá para obtener las interfaces de los objetos y construir la petición de forma dinámica teniendo en cuenta que debe ser especificada para poderla ejecutar.

CORBA provee 4 interfaces que brindan los servicios que se necesitan para invocar dinámicamente un objeto:

- **CORBA::Object:** Define operaciones que todo objeto CORBA debe soportar.
- **CORBA::Request: D**efine las operaciones sobre un objeto remoto.
- **CORBA::NVList:** Provee métodos que ayudan a construir la lista de parámetros y a manipularla.
- **CORBA::ORB:** Define los métodos del ORB de propósito general.

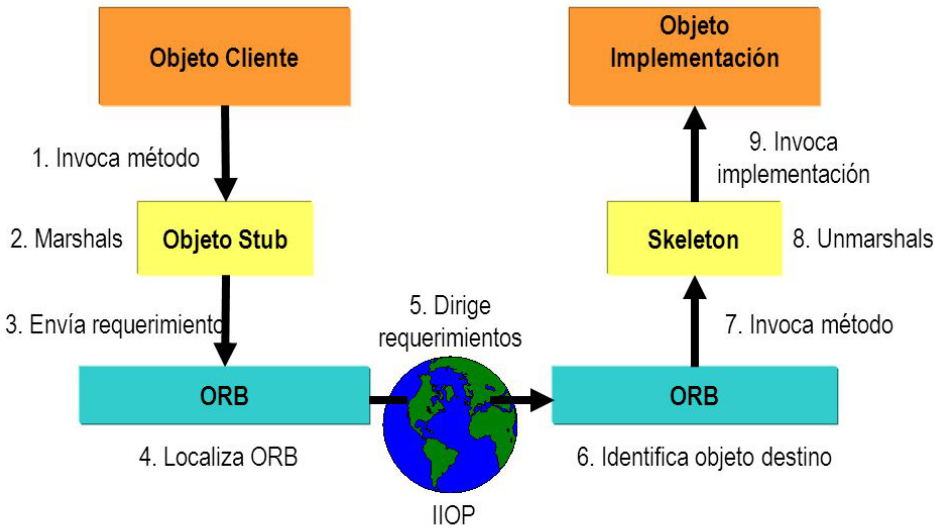

Figura 1.4 CORBA IIOP método de invocación

1.5.3 Esquema Cliente/Servidor 3-Tier

Nuestro sistema de Reservaciones de Vuelo utiliza la arquitectura de tres capas, por lo que vamos a mencionar de qué se trata este esquema. En este esquema la aplicación Cliente/Servidor se divide en tres unidades lógicas funcionales que más tarde puedan asignarse físicamente a un mismo computador o repartirse entre varios. Estas tres unidades lógicas son la interfaz del usuario o lógica de presentación, la lógica del negocio o lógica aplicación y la lógica de acceso a los datos compartidos. Existen muchas variantes posibles de arquitecturas de múltiples dependiendo de la forma en que se divida la

aplicación y del middleware que se emplee para comunicarse entre las capas.

En el caso de los sistemas Cliente/Servidor 2-tier, la lógica de la aplicación está integrada ya sea con la lógica de presentación como parte del proceso cliente o con la lógica de acceso a los datos como parte del proceso servidor. En el caso de los sistemas Cliente/Servidor 3tier (véase figura 1.3) , la lógica de la aplicación ocupa una capa intermedia; que está separada tanto de los datos como de la interfaz del usuario, se les puede administrar y desplegar en forma autónoma, sin relación con la GUI y la base de datos. En teoría, los sistemas Cliente/Servidor de tres capas facilitan más la escalabilidad, suelen ser más robustos y flexibles.

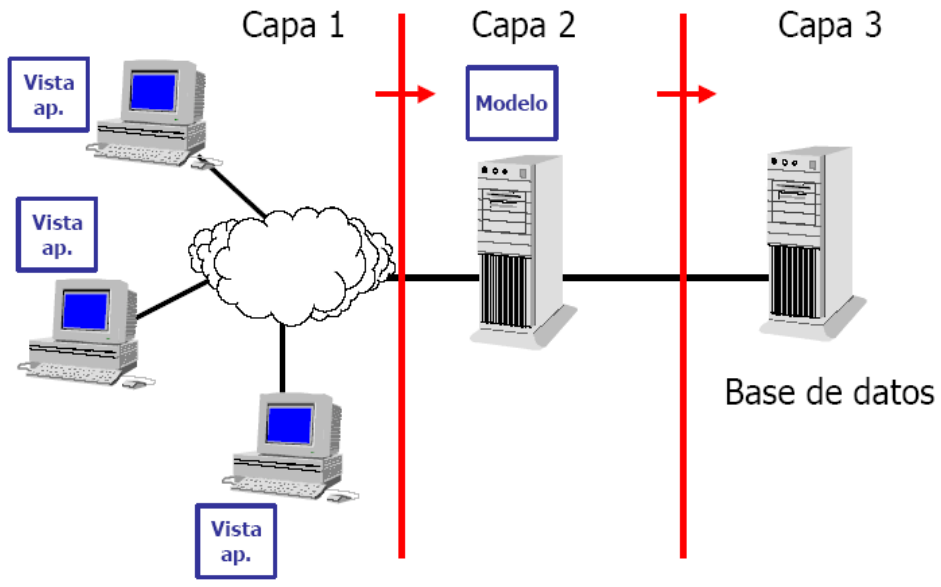

Figura 1.5 Esquema Cliente/Servidor 3-Tier

1.5.4 Objetos Distribuidos

Los objetos distribuidos han modificado radicalmente el estilo de desarrollo de sistemas de software. Con esta nueva tecnología la propuesta es poder juntar complejos sistemas cliente/servidor utilizando componentes de software reutilizables como son los objetos. Cualquiera de los objetos puede ser modificado o reemplazado sin afectar al resto de los componentes en el sistema ni su modo de interactuar. Los componentes pueden agruparse como biblioteca de clases cuyas piezas en su totalidad pueden trabajar juntas en una tarea específica.

La tecnología de los objetos distribuidos es idónea para la creación de flexibles sistemas Cliente/Servidor porque los datos y lógica de negocios se encapsulan dentro de los objetos. De esta manera se pueden ubicar en cualquier punto de la red en un sistema distribuido. Los objetos distribuidos pueden ser conectados y usados, interoperar entre redes, correr en diferentes plataformas y administrarse solos lo mismo que a los recursos que controlan.

Los objetos distribuidos no bastan por sí solos para el cumplimiento de ese objetivo. Tienen que ser agrupados y ser capaces de operar juntos. Para que los objetos operen de acuerdo a lo esperado, deben residir en entornos Cliente/Servidor abiertos y se debe saber cómo conectarlos y usarlos entre las diferentes redes y sistemas operativos.

1.5.4.1 Los Componentes

Un objeto "clásico", no distribuido, es una entidad inteligente que encapsula código y datos, proporcionando facilidades de reutilización de código por medio de herencia y encapsulamiento. Lamentablemente viven en un solo programa. Sólo el compilador del lenguaje que crea los objetos sabe de la existencia de dichos objetos, nadie más sabe de estos objetos y no hay manera de acceder a ellos.

Por otra parte, un objeto distribuido es un ente inteligente que es capaz de vivir en cualquier parte de una red. Los objetos distribuidos se empaquetan como piezas de código independiente a las que pueden acceder procesos clientes remotos por medio de la invocación de métodos. El lenguaje de programación y el compilador utilizado para crear un objeto distribuido es completamente transparente al usuario. Los usuarios ignoran dónde reside físicamente el objeto distribuido o en qué sistema se ejecuta, el objeto distribuido puede estar en la misma máquina o en algún lugar de la red.

Cuando nos referimos a objetos distribuidos, en realidad hablamos de componentes de software independientes. Un componente es un objeto no ligado a ningún programa, lenguaje de programación o implementación en particular. Los objetos creados como componentes son esenciales para las aplicaciones distribuidas. En sistemas de objetos distribuidos, la unidad de trabajo y distribución es el componente. La tecnología de componentes transformará la forma de desarrollar complejos sistemas cliente/servidor y

se considera que la siguiente generación de sistemas distribuidos estará basada en componentes.

1.5.4.2 ¿Por qué usar Componentes?

La programación orientada a objetos no ofrece por sí sola una infraestructura para que el software creado por diversos fabricantes puedan interactuar en el mismo espacio de direccionamiento, y mucho menos entre diferentes espacios de direccionamiento, redes y sistemas operativos. La solución es completar los objetos "clásicos" con una infraestructura de componentes estándar. CORBA de la OMG provee un estándar de componentes para la empresa. DCOM de Microsoft proporciona otro. El impacto de esta tecnología hará que los usuarios encuentren con ésta la mejor manera de ensamblar sus aplicaciones personalizadas haciendo uso de componentes.

Los pequeños desarrolladores y los vendedores independientes de software descubrirán que los componentes reducen costos y derriban las barreras de acceso al mercado de software. Podrán crear componentes individuales con la certeza de que se integrarán armónicamente con et software existente creado por grandes compañías de desarrollo; no tendrán que reinventar todas las funciones.

Los grandes desarrolladores, fabricantes independientes de software e integradores de sistemas usarán series de componentes para crear o ensamblar aplicaciones Cliente/Servidor de alcance empresarial. Lo

común será que cerca del 80% de las funciones que necesiten estén a su disposición bajo la forma de componentes comerciales. El 2O o/o restante será el valor agregado que aporten. La prueba de los sistemas Cliente/Servidor construidos será menos compleja, gracias a la alta confiabilidad de componentes probados previamente. Los profesionales de la comercialización usarán componentes para ensamblar aplicaciones dirigidas a mercados específicos, en vez de vender series demasiado grandes a precios prohibitivos. La mayor personalización de los productos hará surgir nuevas plazas de mercado.

1.5.4.3 ¿Qué es un Componente?

Los componentes interoperan mediante el uso de modelos de interacción Cliente/Servidor estándar en lo que se refiere a mensajes. A diferencia de los objetos tradicionales, pueden interoperar entre lenguajes, herramientas, sistemas operativos y redes. Sin embargo, también son semejantes a los objetos clásicos en el sentido de que soportan herencia, polimorfismo y encapsulamiento. A continuación se define las mínimas propiedades de un componente:

- Es una entidad comercializable. Un componente es una pieza binaria de software autónoma en paquete comercial que puede adquirirse en el mercado abierto.
- No es una aplicación completa. Un componente puede combinarse con otros componentes para formar una aplicación completa. Está

diseñado para realizar un conjunto limitado de tareas dentro del dominio de una aplicación.

- Se le puede utilizar en combinaciones impredecibles. Como los objetos clásicos, un componente puede usarse en formas absolutamente imprevistas por el desarrollador. Usualmente, los componentes pueden combinarse con otros componentes mediante conectar y usar.
- Tiene una interfaz claramente definida. Como los objetos clásicos, un componente sólo puede ser manipulado a través de su interfaz. Este es el medio por el cual el componente expone su función al mundo exterior.
- Es un objeto interoperable. Un componente puede ser invocado como un objeto entre diferentes espacios de direccionamiento, redes de computadores, lenguajes de programación, sistemas operativos y herramientas de desarrollo. Es una entidad de software completamente independiente.
- Es un objeto extendido. Los componentes son objetos auténticos en el sentido de que soportan encapsulamiento, herencia y polimorfismo. Sin embargo, también deben contar con todas las características asociadas con un objeto comercial autónomo y reutilizable.

1.5.4.4 ¿Qué es un Supercomponente?

Los supercomponentes son componentes con facultades adicionales. Estas facultades son necesarias para la creación de objetos comerciales autónomos de acoplamiento holgado capaces de deambular entre máquinas y vivir en redes. Facilidades de los componentes:

Seguridad: Un componente debe protegerse y proteger a sus recursos contra amenazas externas. Debe autenticarse para sus clientes y viceversa. Debe brindar controles de acceso. Además, debe mantener rastros para auditoria de su uso.

Licencias: Un componente debe imponer políticas de licencias, como licencias y medición por uso de sus componentes.

Uso de versiones: Un componente debe suministrar alguna modalidad de control de versiones. Debe garantizarles a sus clientes que usan la versión correcta.

Administración del ciclo de vida: Un componente debe administrar su creación, destrucción y archivamiento. También debe ser capaz de reproducirse, exteriorizar su contenido y desplazarse de un lugar a otro.

Soporte de paletas de herramientas abiertas: Un componente debe permitir su importación dentro de una paleta de herramientas estándar. Un componente que cumple las reglas de paleta abierta puede ensamblase con otros componentes con "arrastrar y soltar" y otras técnicas de ensamble visual.

Notificación de eventos: Un componente debe ser capaza de notificar a las partes interesadas la ocurrencia de algo que les incumba.

Configuración y administración de propiedad: Un componente debe proporcionar una interfaz que le permita a usted configurar sus propiedades.

Metadatos e introspección: Un componente debe ofrecer, a solicitud expresa, información sobre sí mismo. Esto incluye una descripción de sus interfaces, atributos y comportamiento.

Control de transacciones y candados: Un componente debe proteger sus recursos en transacciones y cooperar con otros componentes en el ofrecimiento de integridad total o nula. Adicionalmente, debe ofrecer candados para serializar el acceso a recursos compartidos.

Persistencia: Un componente debe ser capaz de guardar su estado en almacenamiento persistente y de restaurarlo más tarde.

Relaciones: Un componente debe estar en condiciones de asociarse dinámicamente o permanentemente con otros componentes.

Facilidad de uso: Un componente debe ofrecer un número limitado de operaciones para alentar su uso y reutilización.

Autocomprobación: Un componente debe probarse por sí mismo. Usted debe estar en condiciones de correr diagnósticos provistos por el componente para la determinación de problemas.

Autoinstalación: Un componente debe estar en condiciones de instalarse sólo y de registrar automáticamente su entrada en funciones en el registro del sistema operativo o de componentes. También debe eliminarse del disco cuando se le solicite.

1.5.5. Los Objetos de Negocios

Los objetos distribuidos son objetos por definición. La infraestructura de los objetos distribuidos es en realidad una infraestructura de componentes. Los programadores pueden lograr fácilmente una aptitud de colaboración generando código para las dos partes involucradas, lo difícil es conseguir que los componentes sin conocimiento previo entre sí hagan lo mismo. Para llegar a este punto se necesitan estándares que fijen las reglas de participación para diferentes fronteras de interacción entre componentes. En el nivel básico, una infraestructura de componentes aporta un bus de objetos, el intermediario de solicitudes de objetos (ORB: object request broker), el cual permite que los objetos interoperen entre diferentes espacios de direccionamiento, lenguajes de programación, sistemas operativos y redes de computadores. El bus también ofrece mecanismos para que los componentes intercambien metadatos y se descubran entre sí. En el siguiente nivel, la infraestructura complementa el bus con servicios adicionales, los cuales hacen posible la creación de componentes súper inteligentes.

La última meta es permitir la creación de componentes que se comporten como objetos de negocios. Por lo general éstos desempeñan funciones de negocios específicas: un cliente, automóvil, hotel, etc.

Los objetos de negocios son ideales para la creación de soluciones Cliente/Servidor de tres capas fácilmente escalables, porque son inherentemente desarmables. Los objetos de negocios se pueden desarmar y volverse a armar.

Los objetos de la capa intermedia interactúan con sus procesos clientes e implementan la lógica de negocios. Los clientes jamás deben interactuar directamente con las fuentes de datos que se encuentran en la tercera capa. Estas deben ser totalmente encapsuladas y abstraídas por los objetos servidores de la capa intermedia. Los clientes suelen interactuar con los objetos del servidor de la capa intermedia a través de un ORB. Además, los objetos de la capa intermedia pueden comunicarse entre sí por medio de un ORB del servidor que les permita equilibrar cargas, dirigir transacciones distribuidas e intercambiar eventos de negocios. Esto vuelve muy ampliable a los objetos de negocios basados en ORB. Los objetos servidores se comunican con la tercera capa mediante el middleware tradicional.

1.5.6 Procesos Servidores de Objetos

En un sistema Cliente/Servidor existen procesos que solicitan servicios y procesos que los proveen. Dependiendo del servicio que se brinde, existe una clasificación de tipos de procesos servidores. Esto se mencionó en la definición formal que se dio anteriormente cuando se dijo ". . . procesos servidores generalmente especializados... ". Una clasificación de tipos de procesos servidores sería:

File servers, DataBase Servers,. Transaction Processing Servers, GroupWare Servers, Web Servers, Object Servers.

El proceso servidor del Sistema de Reservaciones de Vuelo es de tipo Object Server, es decir, un servidor de objetos. Con un servidor de objetos, la aplicación Cliente/Servidor hace uso de la nueva tecnología de objetos distribuidos. El proceso cliente se comunica con los objetos del servidor mediante un intermediario de solicitudes de objetos (ORB).

El modo de funcionamiento es básicamente éste: El proceso cliente invoca un método de un objeto remoto. El ORB localiza una instancia de ese tipo de objeto, invoca el método solicitado y envía los resultados al proceso cliente. Los objetos del proceso servidor deben ofrecer un soporte de concurrencia y participación. El ORB se encarga de reunir todos los elementos.

1.5.7 Procesos Cliente y Servidor

El proceso servidor y el proceso cliente deberán ejecutarse en computadoras cuyo sistema operativo es Windows. La comunicación entre ambos se realizará utilizando el mejor middleware de nuestra industria actual: los ORB (Object Request Broker) de CORBA y además un pool de objetos de servidor será implementado para encargarse de atender los requerimientos de cada proceso cliente (Figura 1.6). El lenguaje en que han sido implementados tanto el proceso cliente como el proceso servidor, es el lenguaje de programación Java.

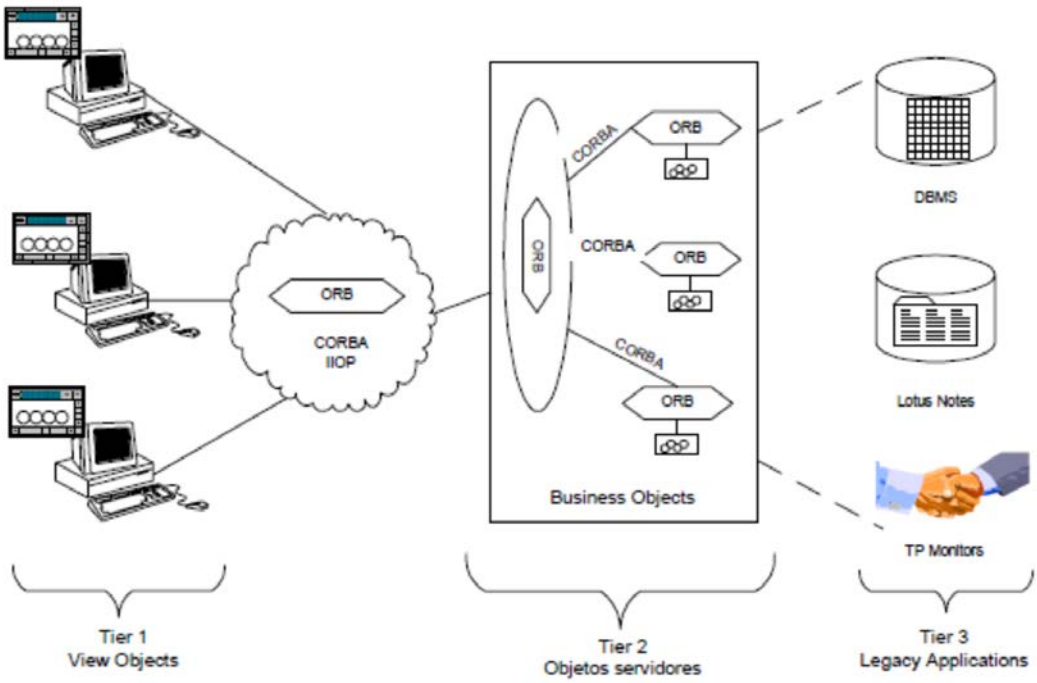

Figura 1.6 Arquitectura del proyecto con CORBA three tier

Ahora detallamos la interacción del proceso cliente con el proceso servidor.

1.5.8 Interacción Cliente-Servidor

Se recomienda que en el computador donde se encuentre ejecutando el proceso servidor también se encuentre el servidor de base de datos junto con la base de datos, ya que así evitamos que tanto datos como sentencias SQL viajen a lo largo de la red, disminuyendo el tráfico en la misma y ayudando en el rendimiento general de la aplicación en lo que a tiempos de respuesta se refiere.

¿Cómo se comunica el proceso cliente con el proceso servidor?

Cuando el proceso servidor se "levanta", se instancia un objeto de tipo ManejaPoolServerObjectsVuelos que es el que implementa el pool de objetos de servidor. Cuando eso ocurre este objeto instancia un número n (n es configurable y se pasa como parámetro al proceso servidor en el momento de su ejecución) de objetos de tipo MMVuelos pre-levantados. El objeto de tipo MMVuelos, como parte de la lógica de su método constructor, instancia cinco objetos de tipo Cliente, Recorrido, Reservación, Ubicación y Vuelo. Estos cinco tipos de objetos son los que permiten al proceso cliente que manipule toda la información correspondiente a todos estos temas y sus referencias son parte de las propiedades del objeto de tipo MMVuelos. Después de esto, el proceso servidor indica que éstos objetos están listos para ser empleados por los procesos clientes simplemente exportándolos al ORB. Lo primero que debe hacer el proceso cliente es tomar la referencia del objeto que implementa el pool, simplemente preguntando por su nombre. Una vez que tiene su referencia puede invocar sus métodos. El primer método que debe invocar es ReservarMMVuelosObject( ) que retorna un objeto de tipo MMVuelos. Con esta nueva referencia puede ahora invocar los métodos de un objeto de este tipo para finalmente tener las referencias de los objetos de tipo Cliente, Recorrido, Reservación, Ubicación y Vuelo que son los que finalmente usará. Una vez que el proceso cliente termina su ejecución este llama al método LiberarMMVuelosObject( ) del objeto que implementa el pool para entregar el objeto de tipo MMVuelos al mismo y que

puede brindar sus servicios a otro proceso cliente que lo requiera. Si el número de procesos clientes concurrentes supera el valor de n, el proceso servidor empezará a instanciar de manera temporal objetos de tipo MMVuelos que una vez que finalicen serán sacados de memoria.

¿Cómo envía el proceso servidor los resultados?

Para el proceso cliente es completamente transparente dónde están localizados los objetos que este utiliza, ya que lo único que hace es invocar métodos de objetos como si estos fueran locales, es decir, como si se encontraran dentro de su mismo espacio de direccionamiento. Es en este momento cuando los ORB redireccionan el requerimiento al objeto que se encuentra en otro espacio de direccionamiento o en otro punto de la red. Entonces el proceso servidor, por medio de los objetos que instancia, devuelve los resultados al proceso cliente como simples llamadas a funciones locales.

1.5.9 Middleware

El middleware es todo lo que está entre los procesos clientes y los procesos servidores, es decir, es básicamente lo que permite la comunicación entre ellos.

Para la mayor parte de la industria de la computación el más importante y ambicioso proyecto de middleware es CORBA (Common Object Request Broker Arquitecture), y ha sido el usado por nuestro sistema. Sin embargo

existe una clasificación para el middleware que debemos mencionar. Tres formas o componentes:

- Mecanismos de Comunicación
- Funciones especiales de los Sistema Operativos de Red (NOS)
- Middleware para Servicios Específicos

Uno de los factores que hace importante un middleware es la transparencia. La transparencia significa hacerle creer a la gente que el sistema Cliente/Servidor es único. Significa ocultarles a los usuarios e incluso a los programadores de aplicaciones tanto la red como su servicio.

CAPÍTULO 2

2 RESULTADOS OBTENIDOS

2.1 Diseño de Interfaces Gráficas de Usuario

Se analizó varias posibilidades de cómo hacer el cliente, ya que podría ser de varios applets sencillos que contengan una transacción a la vez. Pero se tuvo una dificultad con este tipo de cliente. Otra posibilidad fue la que hemos escogido. Se lo realizo con el enfoque de un Applet que pueda contener varios paneles y de esta manera superar el inconveniente de poder mantener información de una transacción anterior de un cliente sin tener la necesidad de accesarlo otra vez. El applet del Cliente fue realizado con la ayudad de Visual Café permitiendo tener un Tab de paneles para todas las opciones de transacción. El cliente solo con dar un click del ratón sobre botones en la interfaz gráfica llama a funciones que se encuentran en objetos ubicados en el servidor. Por medio de la interface se accesa a estos objetos. El código de la interacción con los objetos Corba no

presenta ninguna dificultad para el programador ya que se trata a los objetos servidores como si fueran objetos locales. En el Applet se realizaron varios mensajes de Error que sirven para evitar realizar las transacciones en forma incorrecta. Estos mensajes de errores, en su mayoría no llaman a ningún objeto sino que comprueba su veracidad dentro del propio Applet. Los errores que detecta son de formato, nombre, incompletos, campos nulos, etc. Los Paneles que contiene este Applet son de reserva, cancelación de reservas, ingreso y cancelación de vuelos; ingreso de recorridos, ingreso de ubicaciones y consultas.

Applet Principal es el nombre que se le ha dado al applet, encargado de facilitar al Cliente una interfaz gráfica para el manejo de las transacciones en una manera amigable. Es el encargado de llamar a la función que nos conecta con el servidor atravez de su interface ORB. La Interfaz gráfica consta de 2 partes que se encuentran dentro de un applet La primera es un panel que hemos denominado Principal, La segunda es el TAB panel, el mismo que contiene seis Paneles y consta de las seis transacciones

2.1.1 Registro de clientes

El cliente deberá escoger la opción de registro (Figura 2.1) e ingresar la información que se lo solicita: Apellidos, nombres, usuario, clave, país, ciudad, dirección y su e-mail. Una vez introducidos todos estos datos, el cliente deberá hacer “click” en el botón Registrar.

El sistema en la barra de status, le indicara si el proceso de registro fue exitoso o si hubo problemas

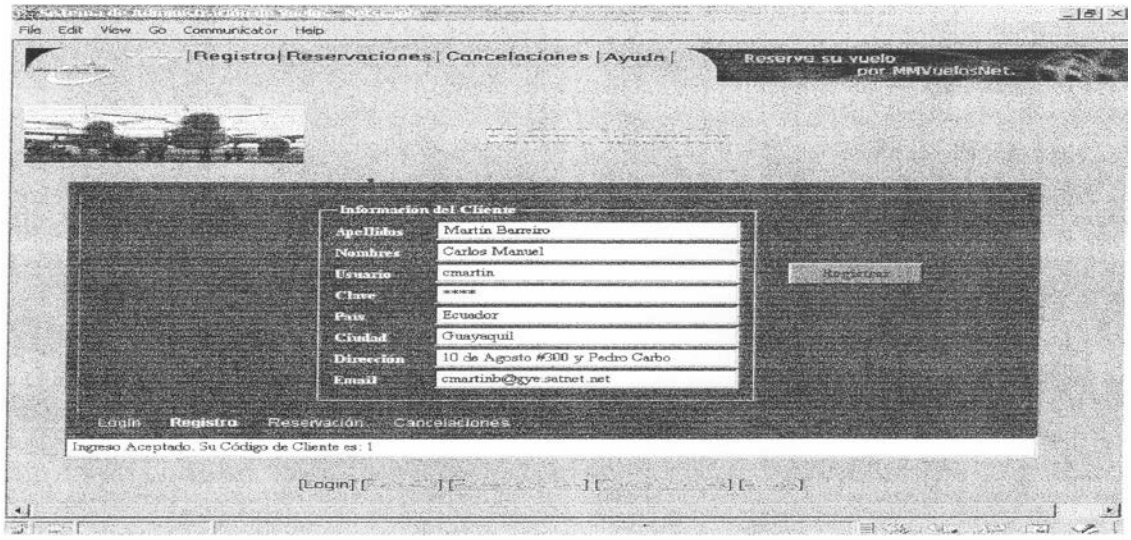

Figura 2.1 Registro de clientes

2.1.2 Autenticación

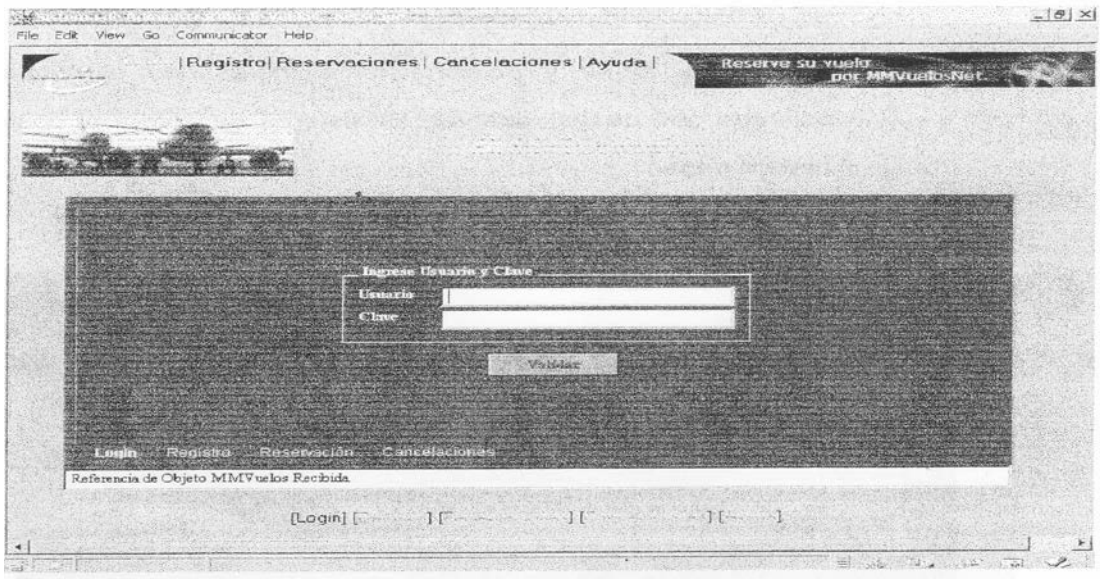

Figura 2.2 Autenticación

Si el cliente ha usado el sistema en anteriores oportunidades, deberá estar registrado (Figura 2.2), por lo que puede ingresar al mismo haciendo ·log _in". De igual manera indicará en la barra de status si la autenticación fue exitosa reconociendo al usuario del sistema mediante su información de usuario y clave.

2.1.3 Reservaciones de vuelo

Podrá realizar las respectivas reservaciones escogiendo el vuelo que desee tomar y adicionalmente ingresando el número de asientos que debe reservar (Figura 2.3). El sistema devolverá el costo que deberá pegar.

El cliente debe seleccionar el vuelo a tomar e ingresar el número de reservas.

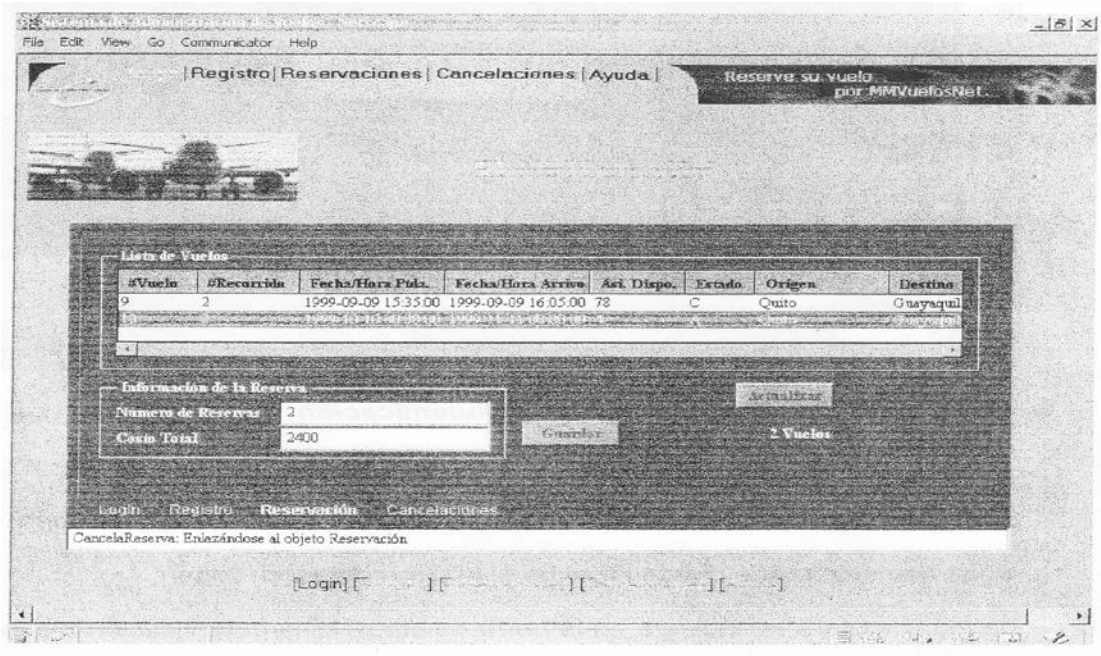

Figura 2.3 Reservas de Vuelo

2.1.4 Cancelación de Reversas de Vuelo

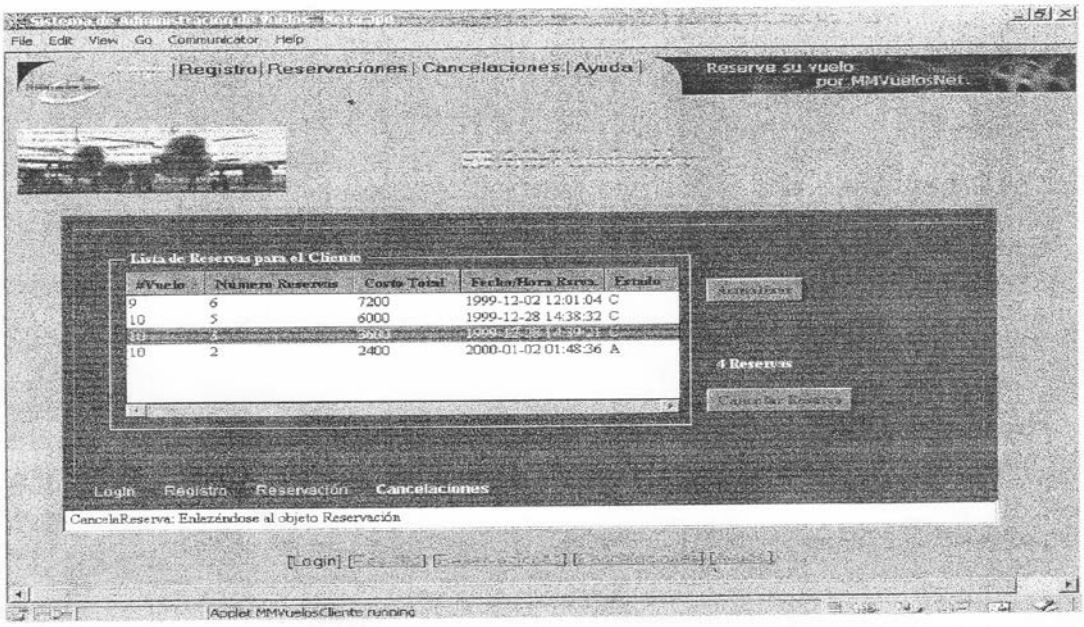

Figura 2.4 Cancelación de Reservas de Vuelo

Existe la opción de poder cancelar las reservaciones (Figura 2.4). El usuario deberá seleccionar la reserva, dentro de la lista de reservas para el cliente, y luego cancelaria presionando el botón “Cancelar Reserva”

El sistema indicará al usuario, mediante la barra de status, el resultado de la cancelación de la reserva de vuelo

2.2 Mejoras en el sistema

A continuación se detallarán las mejoras de implementar el sistema de reservaciones de vuelo:

- **Permitir al administrador y a los clientes:** Manejar el sistema con una interfaz gráfica fácil de usar:
- **Agilita las reservaciones:** El sistema permite que los clientes de la aerolínea reserven sus vuelos mediante el web sin necesidad de acercarse físicamente a las oficinas de la empresa.
- **Información en línea:** El sistema brinda toda la información tal como vuelos recorridos y ubicaciones de tal forma que los usuarios del web siempre estén actualizados de cualquier nueva actividad
- **Privacidad al hacer las Reservaciones:** Cada cliente tiene su propio identificador, de manera tal que nunca un cliente puede consultar o cancelar las reservaciones de otros clientes.
- **Clientes Concurrentes:** Debido a que el proceso servidor implementa un "Pool de objetos de Servidor", el sistema tiene la capacidad de atender a un ¨"ilimitado" número de procesos clientes concurrentes. La única limitación del sistema es todo el ambiente de hardware en el que se desenvuelve
- **Fácil y eficiente revisión de Reservaciones por parte del Administrador:** El administrador podrá revisar las reservaciones de todos los clientes y verificar su estado, de tal manera que se podrá detectar quienes son los que más reservaciones hacen y con qué frecuencia cancelan sus reservas

2.3 Limitaciones

- No existe la opción de escoger el número de asiento que el pasajero desea al momento de hacer reservas, además todos los asientos tienen igual jerarquía, es decir, no existe primera clase, segunda clase, tercera clase, etc.

- Si un cliente hace más de una reservación, todas las reservaciones son asignadas a él y no se permite que se ingresen los nombres de las otras personas que van a viajar con él, en ese caso tendría que físicamente acercarse y dar sus nombres a la hora de pagar sus boletos y retirarlos.
- El sistema no permite vuelos compuestos, es decir, vuelos con más de un segmento entre destinos finales
- el costo para un vuelo es fijo, lo que significa que no varían de acuerdo a las temporadas. Esto se menciona debido a que las aerolíneas reales brindan unos costos en temporada alta y otros en temporada baja.

En general existen beneficios de implementar aplicaciones en un sitio Web, hoy en día los usuarios buscan más que información en un sitio Web. Desean tener sistemas a su disposición para satisfacer rápidamente y en cualquier momento alguna necesidad especifica. Por otra parte muchas empresas obtienen grandes beneficios proveyendo estas soluciones a sus clientes y usuarios consiguiendo reducir costos, aumentar ventas, mejorar la imagen de la empresa, conservar clientes, etc. Se debe pensar en la utilidad que ofrecen las aplicaciones web.

Algunas aplicaciones que puede incluir en un sitio Web:

Comercio electrónico

Seguimiento online de operaciones

Consulta de estados de cuenta

Envío masivo de información

Webmail

CONCLUSIONES Y RECOMENDACIONES

Al culminar la fase de diseño del proyecto se pudo constatar lo siguiente:

Conclusiones

1. Para toda la industria de la computación con la notable excepción de Microsoft corporation (en el año 1999) la generación de middleware era CORBA

2. El desarrollo de aplicaciones distribuidas con esquemas 3 tier se facilita cuando empleamos una tecnología que aporte con un bus de objetos y que aplique los conceptos de ORB

3. La computación de internet no reemplaza a la arquitectura cliente/servidor ya que ésta es en sí misma cliente/servidor.

4. Los DBMS cliente/servidor tiene una participación importante dentro de todo el ambiente cliente/servidor en el que se desenvuelve la aplicación distribuida y muchas veces es un componente infaltable

5. Las siguientes generaciones de sistemas distribuidos estarán basados en objetos distribuidos y componentes

6. En un contexto distribuido es donde los objetos brindaran todo su potencial

7. CORBA es una especificación. No es un software o aplicación. Hay un gran número de implementaciones de CORBA. Estas son conocidas como Object Request Broker (ORB).

8 Se nota también que muchas tecnologías están en constante desarrollo y maduración, lo cual implica un minucioso estudio previo de muchos factores antes de apostar por alguna tecnología en especial

9 En todo proyecto es recomendable realizar una correcta planificación, por ende, seleccionar una metodología que permita minimizar los riesgos, aumentar la calidad y acortar el tiempo de implementación ayuda en gran medida al equipo de trabajo.

10 CORBA es hoy en día una opción tecnológica aceptada por la industria del software en lo referente al desarrollo de soluciones distribuidas. A CORBA le surgen competidores, especialmente desde el mundo de Microsoft y desde el mundo de Java, aunque CORBA presenta las ventajas de no ser propietario, no estar ligado a ningún lenguaje de programación concreto y además ser capaces de integrarse sus competidores mediante las pasarelas adecuadas, en un claro ejemplo de interoperabilidad entre sistemas heterogéneos.

11 Java como herramienta también integra interoperabilidad con CORBA siempre y cuando los objetos estén usando un ORB compatible con las especificaciones y que se apoyen con IIOP como protocolo de comunicaciones.

12 El Desarrollo Basado en Componentes es un nuevo paradigma de programación que promete una fácil reutilización de los componentes. Aunque ya hace varios años que salió a la luz, todavía no existe un lenguaje que proporcione un soporte completo para programar sobre él. Esta falta está motivada a que todavía no están completamente definidos algunos conceptos básicos, como son: componente, conector y puerto. Aunque existen algunos lenguajes que implementan funcionalidades reducidas de este paradigma, todavía no existen aplicaciones Drag and Drop que permitan una buena manipulación/interconexión de los componentes.

Recomendaciones

1 Se recomienda explorar y conoce bien el estándar CORBA y sus servicios antes el posible diseño de un proyecto bajo esta herramienta. Esto ahorrará tiempo y problemas.

2 Se recomienda dar a conocer el presente proyecto para futuras investigaciones de implementaciones con CORBA y el lenguaje de programación JAVA

3 Se recomienda continuar con el uso de este estándar CORBA y hacer revisiones periódicas de nuevos especificaciones e implementaciones de componentes

4 La construcción y utilización de clases reutilizables es una meta que los desarrolladores siempre deben tener presente, aunque tampoco se ha de convertir en una obsesión.

5. Durante las fases de planificación y diseño de los sistemas informáticos se debe considerar si el sistema a desarrollar debe o no ser internacionalizado ya que actualmente la demanda de acceso a la información exige romper las limitantes del idioma y así el sistema será más escalable.

6. Tal vez sería interesante, que en un futuro, se profundizara más sobre IDL en este ramo, ya que sin duda CORBA representa un estándar que no se puede dejar de lado.

BIBLIOGRAFÍA

[1] ARNOLD Y GOSLING, El lenguaje de programación Java, Addison – Wesley/Domo, 1997.

[2] MICROSOFT CORPORATION, mastering Distribuited Application Design, Student Workbook Microsoft,1998

[3] ORFALI Y HARKEY Y EDWARDS, Cliente/Servidor Guía de supervivencia, segunda edición, 1997

[4] LEMAY LAURA Y PERKINS CHARLES. Aprendiendo Java 1.1 en 21 días. Segunda edición, Prentice Hall, 1998

[5] 0. GRAF, A. KOTZEN, 0. TAKAGIWA & U. WAHLI, "VisualAge for Java Enterprise Version 2: Data Access Beans - Servlets - CICS Connector", 1era. Edición: New York: Red book de IBM, http://www.redbooks.ibm.com, 1998.

[6] BOOCH GRADY. Análisis y Diseño Orientado a Objetos con Aplicaciones. Segunda edición, Addison-Wesley/Díaz de Santos, 1996.

[7] GRIFFITH & CHAN & F.ISA1, 1001 Tips para Programar con Java, 1era edición en español México, Mc Graw Hill, 1998