ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

Trabajo de Graduación

"Desarrollo de una Aplicación Cliente/Servidor basada en CORBA"

Sistema de Reservaciones de Vuelo para una Aerolínea

Previo a la obtención del título de:

INGENIERO EN COMPUTACIÓN

Según Resolución 2016-14 del Consejo Directivo de la FIEC, resuelve que el único que se gradua fue el Sr. Carlos Manuel Martín Barreiro.

Presentado por:

CARLOS MANUEL MARTÍN BARREIRO

SANDRO HOLGER DELGADO DELGADO

ERIC YOBANNY GUAGUA ALVARADO

Guayaquil – Ecuador

1999

AGRADECIMIENTO

A Dios, por darnos la oportunidad de vivir este momento tan especial,

a nuestros familiares, quienes fueron una motivación constante

a lo largo de nuestras vidas,

a todos los amigos que colaboraron de manera indirecta con el

desarrollo de este proyecto,

y de manera sumamente especial al Ing. Carlos Valero

por toda su ayuda y paciencia.

DEDICATORIA

Dedico este título a Jesús y por medio de Él a Jehová, Mi Dios.

A mi madrecita Rosalía, por todo su amor y su esfuerzo

en ayudarme a salir adelante.

A mi abuelita Alicia, por todo su cariño y sus consejos.

A mi tía Maruja, por ser como una madre para mí.

A mi padre, por transmitirme su experiencia y por sus consejos.

A mis hermanos, a mis demás familiares y a mis verdaderos amigos,

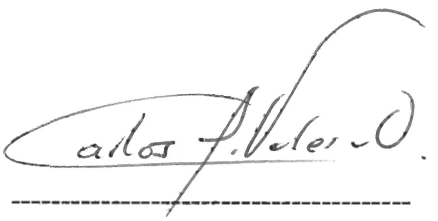
pues siempre me ayudaron y motivaron

A todos, Gracias.

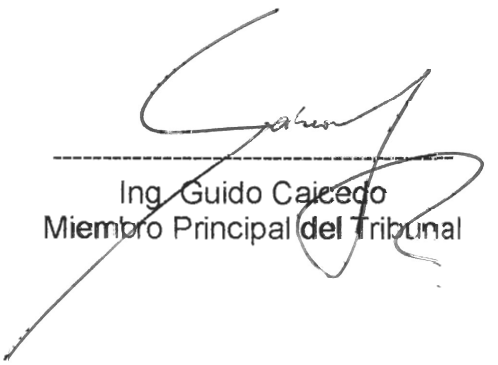
Carlos Manuel Martín Barreiro

MIEMBROS DEL TRIBUNAL

Ing. Carlos Monsalve
Presidente del Tribunal



Ing. Carlos Valero
Director de Tesis



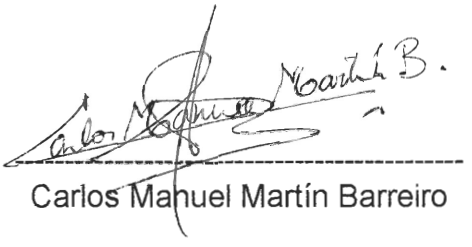
Ing. Guido Caicedo
Miembro Principal del Tribunal

Ing. Rebeca Estrada
Miembro Principal del Tribunal

DECLARACION EXPRESA

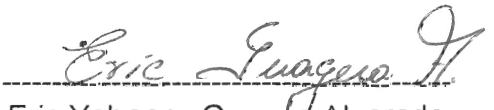
"La responsabilidad por los hechos, ideas y doctrinas expuestos en esta tesis, nos corresponden exclusivamente; y, el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL"

(Reglamento de Exámenes y Títulos Profesionales de la ESPOL).



Carlos Manuel Martín Barreiro

Sandro Holger Delgado Delgado



Eric Yobanny Guagua Alvarado

RESUMEN

Hasta hace unos pocos años crear sistemas cliente/servidor implicaba mucho esfuerzo por parte de los desarrolladores, debido principalmente, a que el middleware no estaba tan desarrollado.

Hoy, cuando estamos a las puertas del nuevo milenio, las cosas son distintas. Ya existe en el mercado del software un middleware muy poderoso. Emplear esta nueva tecnología en la construcción de sistemas distribuidos y demostrar que se facilita el desarrollo cuando están sujetos a la arquitectura de CORBA (Common Object Request Broker Arquitecture) es nuestro principal objetivo. El proyecto realizado involucra el desarrollo de una aplicación que permita la reserva de asientos de vuelo en una empresa de aviación con las siguientes transacciones:

Reserva de Vuelos: El usuario debe primero seleccionar el vuelo que desea tomar y después ingresar el número de asientos que desea reservar. El sistema devolverá el precio del boleto y guardará la información ingresada por el usuario.

Cancelación de Reservas de Vuelo: El usuario selecciona la reserva que desea cancelar y el sistema debe dejar disponibles los respectivos asientos.

Ingreso de Vuelos: Se debe incluir el recorrido que seguirá el avión, el número de asientos disponibles para los pasajeros, el precio del boleto, la fecha y hora de partida del vuelo y la fecha y hora de llegada del mismo.

Cancelación de Vuelos: En este caso, automáticamente también se cancelarán todas las Reservas hechas para ese Vuelo.

Ingreso de Recorridos: El usuario deberá ingresar la ciudad de donde saldrá el vuelo y la ciudad donde llegará, el número de kilómetros a recorrer y el precio de cada boleto.

Ingreso de Ubicaciones: Esta opción permite el ingreso de las respectivas ciudades que serán tomadas en cuenta para los recorridos que seguirán los respectivos aviones.

Consultas: Involucra todo tipo de consulta a las diferentes entidades de la base de datos: Vuelos, Recorridos, Clientes, Ubicaciones y Reservaciones.

ÍNDICE GENERAL

INTRODUCCION

La computación ha evolucionado muchísimo en las dos últimas décadas, tanto a nivel de hardware como a nivel de software.

En cuanto al software, la manera de construir sistemas ha 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 de sistemas 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 ya sabe que los objetos son maravillosos, y que sencillamente no podríamos vivir sin ellos. Existe mucha bibliografía repleta de artículos sobre los prodigios de 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.

Anteriormente, 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?, ¿qué ocurriría si abandonamos el espacio de direccionamiento de un proceso e intentamos que los objetos se comuniquen entre sí a través de una red?

Para decirlo brevemente, ellos necesitaban comprender cómo extender la tecnología de objetos para manejar los complejos asuntos inherentes a la creación de robustos sistemas cliente/servidor.

Ahora, cuando estamos a las puertas del 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 de nuestro proyecto, demostrar que la siguiente generación de sistemas cliente/servidor estará basada en objetos distribuidos.

Estamos profundamente convencidos de que ésta es el área en la que los objetos desarrollarán todo su potencial.

También creemos 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 nos permite hacer uso de objetos distribuidos. Se llama CORBA (Common Object Request Broker Arquitecture), 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.

1. ESPECIFICACIONES

1.1 Objetivos del Proyecto

- El sistema debe ser desarrollado con arquitectura Cliente/Servidor, haciendo uso de un esquema de tres capas.

- Hacer uso de la Tecnología de Objetos Distribuidos y demostrar su importancia para la nueva generación de sistemas con arquitectura Cliente/Servidor.

- Emplear la tecnología que nos ofrece la arquitectura CORBA y demostrar lo conveniente que es para el desarrollo de sistemas Cliente/Servidor de tres capas.

- Demostrar la importante participación que tiene un DBMS Cliente/Servidor en un ambiente de tres capas.

- Comprobar que la arquitectura CORBA es un importante aporte que los desarrolladores de aplicaciones para el Web deben tomar en cuenta al momento de construir sus aplicaciones para Internet.

1.2 Requerimientos funcionales

1.2.1 Requerimientos del Proceso Servidor

- 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.

- Implementar parte de la lógica de la aplicación en stored procedures usando un manejador de bases de datos (DBMS) Cliente/Servidor.

- Emplear CORBA usando los ORB de Inprise VisiBroker como el middleware en nuestro ambiente Cliente/Servidor.

1.2.2 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.

1.2.3 Requerimientos para el Cliente de la Aerolínea

1. Deberá registrarse la primera vez que desee hacer una reservación, las demás veces que desee usar el sistema bastará con que valide su ingreso en la opción de autenticación.

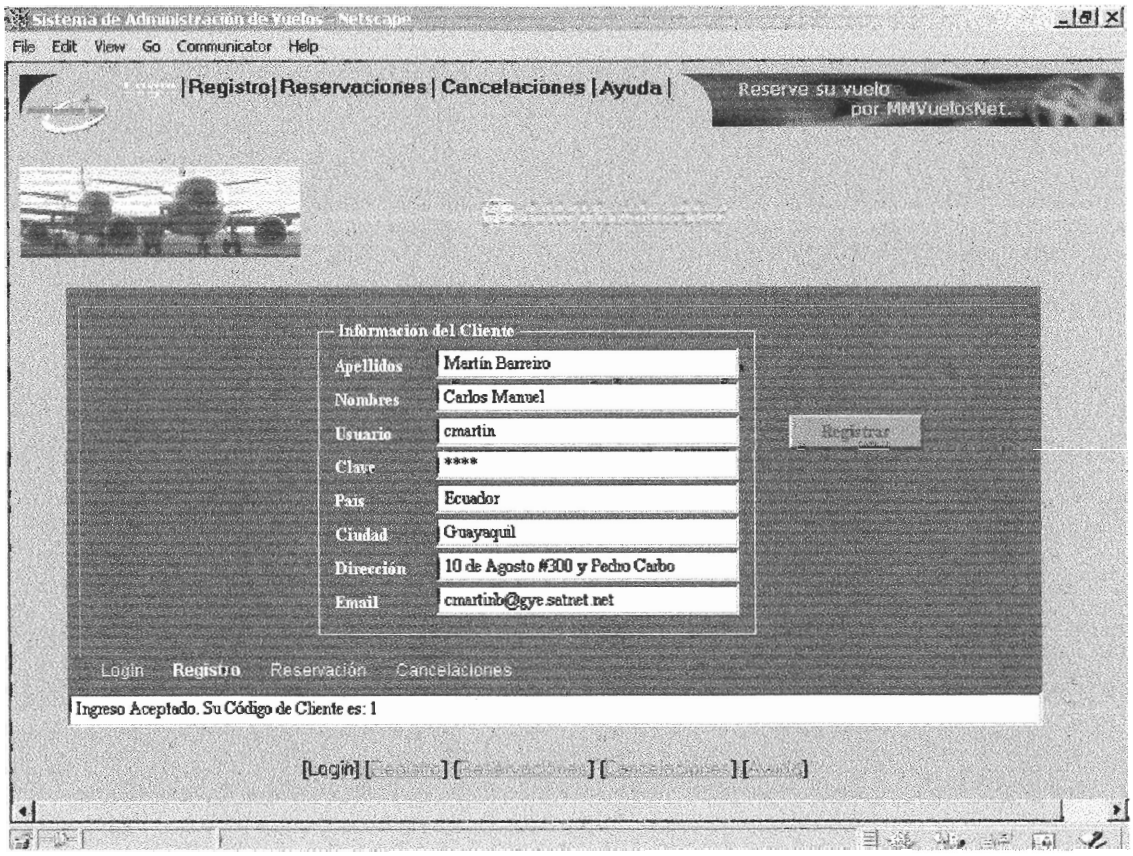


Figura 1.1 Registro de Clientes

El cliente deberá escoger la opción de Registro e ingresar la información que se le 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 indicará si el proceso de registro fue exitoso o si hubieron problemas.

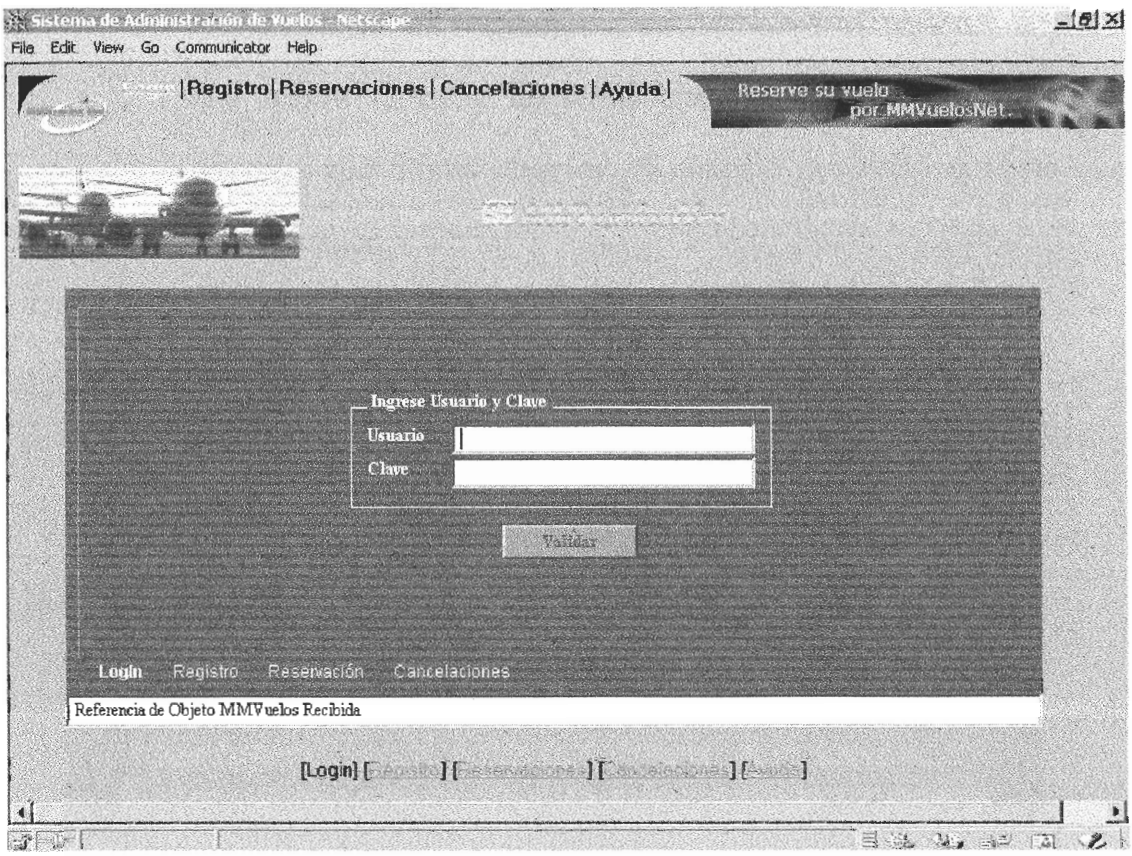


Figura 1.2 Autenticación

Si el cliente ya ha usado el sistema en anteriores oportunidades, deberá estar registrado, por lo que puede ingresar al mismo haciendo 'Log-in'.

De igual manera se indicará en la barra de status si la autenticación fue exitosa reconociendo al usuario del sistema mediante su información de usuario y clave.

2. Podrá realizar las respectivas reservaciones escogiendo el vuelo que desee tomar y adicionalmente ingresando el número de asientos que desea reservar. El sistema devolverá el costo que deberá pagar.

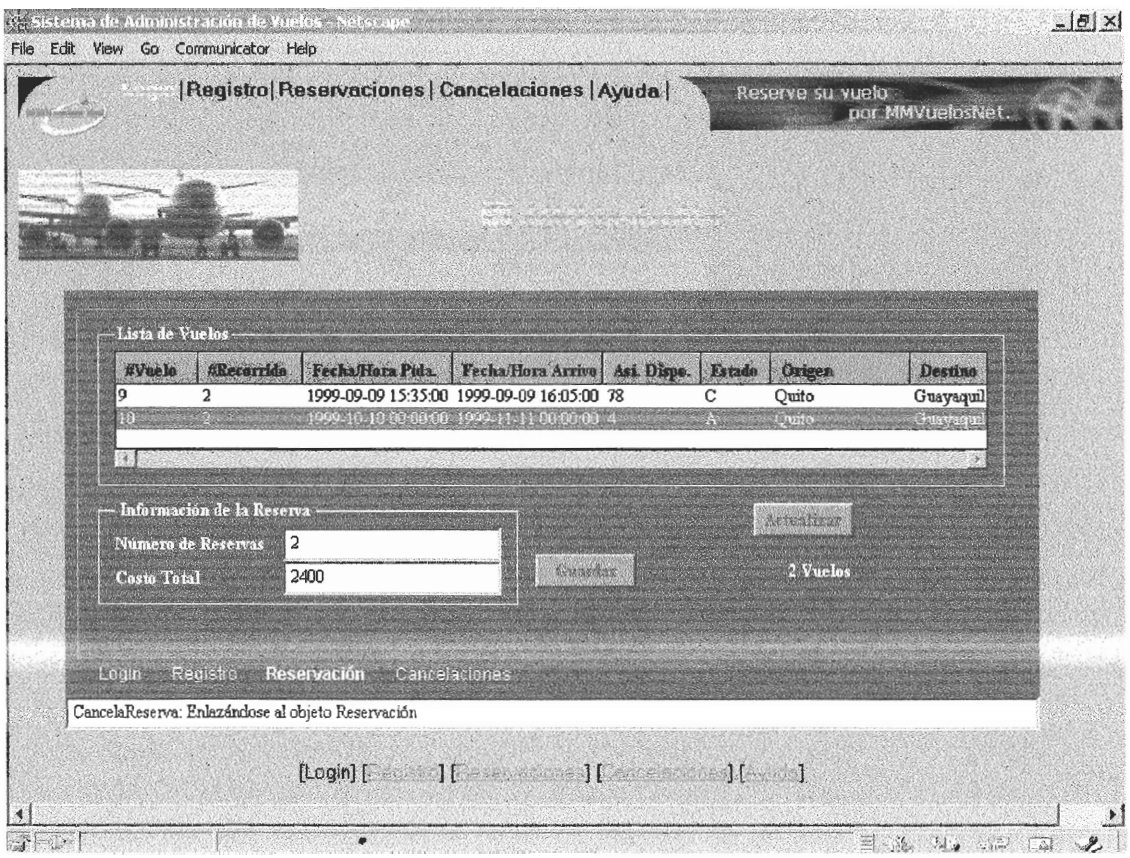


Figura 1.3 Reservaciones de Vuelo

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

3. Existe la opción de poder cancelar las reservaciones.

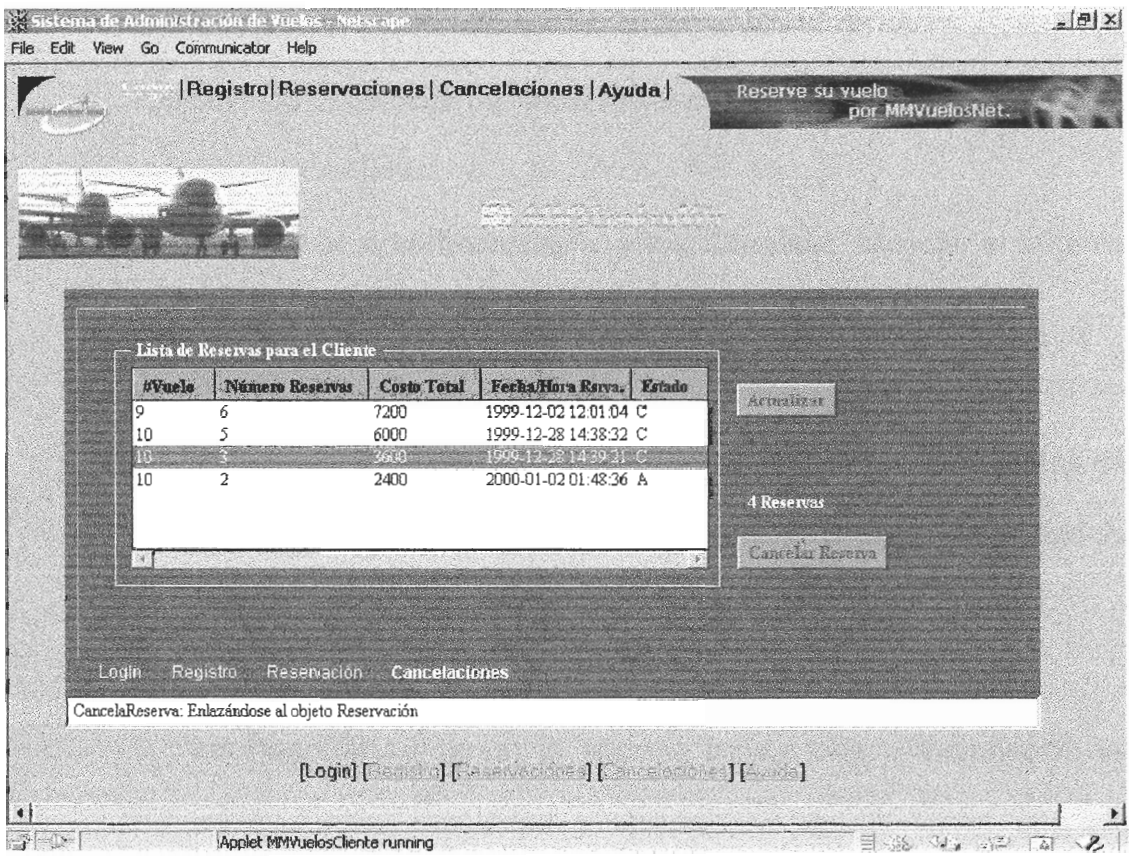


Figura 1.4 Cancelación de Reservas de Vuelo

El usuario deberá seleccionar la reserva, dentro de la lista de reservas para el cliente, y luego cancelarla 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.

1.3 Justificación del Proyecto

A continuación se detallarán las justificaciones de la necesidad 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 quiénes son los que más reservaciones hacen y con que frecuencia cancelan sus reservas.

1.4 Limitaciones

- No existe la opción de escoger el número de asiento que el pasajero desea al momento de hacer la reserva, 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 determinado 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".

1.5 Restricciones del sistema

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

- Utilizar cualquier sistema operativo Windows, pero recomendamos tecnología NT por ser más eficiente y seguro.

- Un "Browser" Java-enabled.

- Un compilador de Java jdk1.1.7

- 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 como mínimo, pero Pentium II 200Mhz recomendable para las computadoras donde van a correr tanto los procesos clientes como el proceso servidor.

Memoria

- 24 MB donde corra el proceso servidor como mínimo, pero recomendable 32MB y 16 MB en aquellas computadoras donde corran los procesos clientes.

1.6 Alcance

El sistema está implementado en Java que es un lenguaje que permite hacer aplicaciones para el 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.

2. ARQUITECTURA DEL SISTEMA

2.1 Arquitectura Cliente/Servidor

El Sistema de Reservaciones de Vuelo posee una arquitectura Cliente/Servidor. 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 gráfico del usuario. Las aplicaciones más avanzadas requieren conocimientos de objetos distribuidos e Internet. Debemos entonces mencionar algo de estos conceptos.

¿Qué es Cliente/Servidor?

A pesar de que Cliente/Servidor es la palabra de moda en la industria de la computación, aún no hay una definición exacta debido a que no ha habido un consenso sobre el significado del término. Esto nos ofrece la magnífica oportunidad de proponer la definición aprendida en el tópico de graduación. Como se deduce del término mismo, 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"

Ing. Carlos Valero

Creemos que todos los sistemas Cliente/Servidor poseen las siguientes características:

- Servicio: Cliente/Servidor es fundamentalmente una relación entre procesos. El proceso servidor hace de éste un proveedor de servicios. El proceso cliente es un consumidor de servicios. En esencia, Cliente/Servidor aporta una clara distinción de funciones con base en la idea de servicio.

- Recursos Compartidos: Un proceso servidor puede atender a muchos clientes al mismo tiempo y regular su acceso a recursos compartidos.

- Protocolos Asimétricos: Entre procesos clientes y proceso servidor se establece una relación de muchos a uno. Son siempre los clientes quienes inician el dialogo al solicitar un servicio. Los procesos servidores aguardan pasivamente las solicitudes de los clientes.

- Transparencia de ubicación: El proceso servidor puede residir en el mismo equipo de cómputo que el cliente o en uno distinto a lo largo de una red. El proceso servidor suele ocultarles a los procesos clientes su ubicación mediante el redireccionamiento de las llamadas de servicio en caso necesario. Un proceso puede ser un cliente, un servidor o ambos.

- Mezcla e igualdad de Plataformas: El software ideal Cliente/Servidor es independiente del hardware o de las plataformas de software como el sistema operativo. Usted debe estar en condiciones de mezclar e igualar plataformas de procesos cliente y servidor.

- Intercambios basados en mensajes: Procesos clientes y servidores son sistemas holgadamente acoplados que interactúan a través de un mecanismo de transmisión de mensajes. El mensaje es el mecanismo de entrega para las solicitudes y respuestas de servicio.

- Encapsulamiento de servicios: El proceso servidor es un "especialista". Un mensaje le indica a un proceso servidor qué servicio se solicita; éste se le envía luego al proceso cliente para determinar el cumplimiento de la tarea. Los servidores pueden ser sustituidos sin afectar a los clientes, siempre y cuando la interfaz para la publicación del mensaje no cambie.

- Facilidad de escalabilidad: Los sistemas de Cliente/Servidor pueden escalarse horizontal o verticalmente. La escalabilidad horizontal significa la adición o eliminación de procesos clientes con apenas un ligero impacto en el desempeño. La escalabilidad vertical significa migrar a un computador con más recursos y más veloz o a computadores múltiples.

- Integridad: El código del proceso servidor y los datos del servidor se conservan centralmente, lo que resulta en un mantenimiento de menor costo y en la protección de la integridad de los datos compartidos. Al mismo tiempo, los procesos clientes mantienen su individualidad e independencia.

Estas características permiten la fácil distribución de inteligencia y procesamiento a lo largo de una red.

2.1.1 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).

2.1.3 Procesos Servidores de Bases de Datos

Tal vez el más fundamental e importante de todos los tipos de procesos servidores. También es un producto listo para instalar y usar. Podría decirse que muchas veces es un "componente infaltable" dentro de un ambiente Cliente/Servidor ya que es el responsable de todos los datos y toda la información.

Recibe requerimientos de los procesos clientes, típicamente en forma de SQL calls, es decir, usando un lenguaje especial conocido como SQL (Lenguaje estructurado de consulta).

Es mencionado porque nuestro sistema hace uso de un servidor de bases de datos conocido, llamado SQL Server 7.0, producto de Microsoft Corporation. Es en este producto donde está implementada la base de datos.

2.1.4 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 de la aplicación y la lógica de acceso a los datos compartidos.

Existen muchas variantes posibles de arquitecturas de múltiples capas, dependiendo de la forma en que se divida la aplicación y del middleware que se emplee para comunicarse entre las capas.

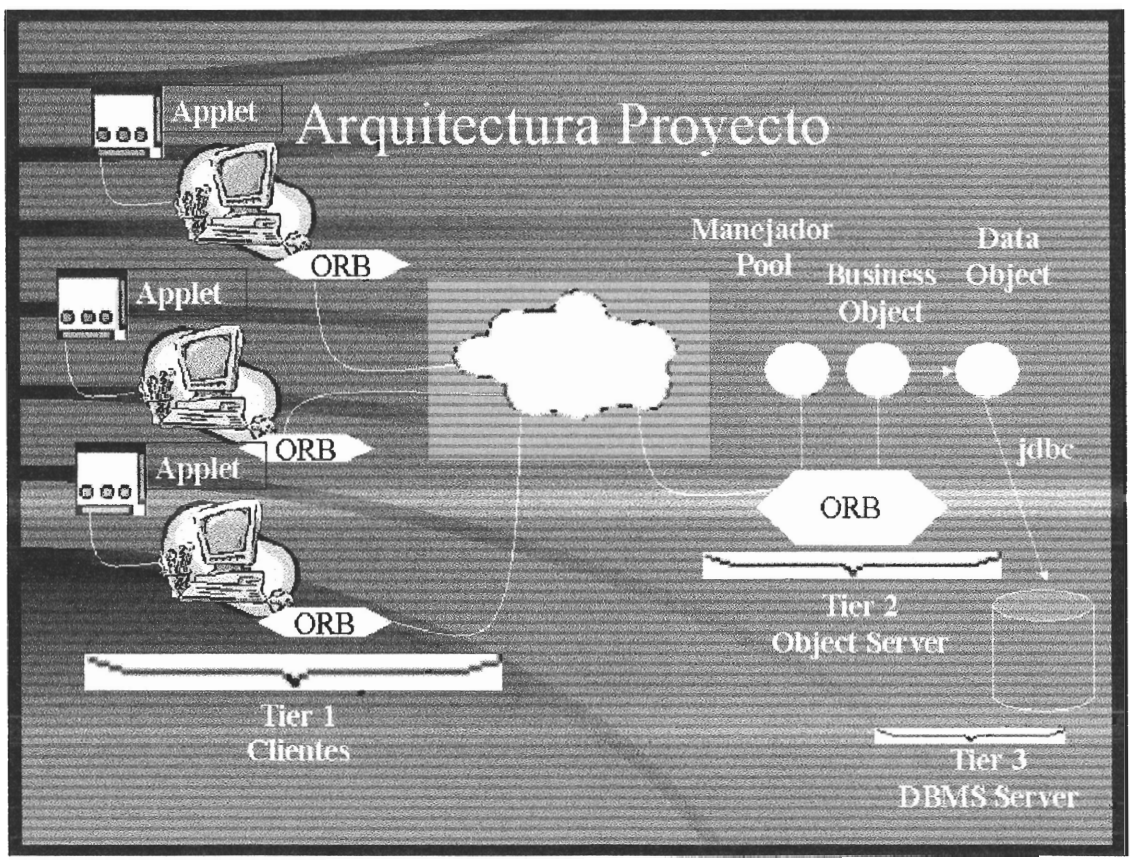


Figura 2.1 Arquitectura del Proyecto

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 3-tier, la lógica de la aplicación ocupa una capa intermedia; que está separada tanto de los datos como de la interfaz del usuario. Los procesos que implementan la lógica de negocios se convierten en ciudadanos de primera clase; 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.

Para el sistema de reservaciones de vuelos, la capa uno son applets de java que corren dentro de un browser, todo el middleware es CORBA, la capa de la mitad es el servidor de objetos o CORBA Server y la última capa es la formada por el sistema manejador de base de datos o DBMS. El acceso a la base de datos es mediante drivers jdbc y algo importante que hay que mencionar es que la mayor parte de la lógica de aplicación está implementada mediante stored procedures de SQL Server de Microsoft Corporation.

2.2 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 lo 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 como conectarlos y usarlos entre las diferentes redes y sistemas operativos.

2.2.1 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.

¿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 el 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 20% 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.

¿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. Esta 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.

¿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 nos permita 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.

2.2.2 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 superinteligentes.

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.

2.3 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. El lenguaje en que han sido implementados tanto el proceso cliente como el proceso servidor, es el lenguaje de programación Java.

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

2.3.1 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.

2.4 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.

¿Qué significa transparencia?

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. A continuación mencionamos algunos tipos de transparencia:

- Transparencia de ubicación: Usted debe conocer la ubicación de un recurso. Los usuarios no deben verse obligados a incluir la información de ubicación en el nombre del recurso.
- Transparencia de espacio para nombres: Usted debe estar en posibilidad de emplear las mismas convenciones de nombramiento para localizar cualquier recurso en la red.
- Transparencia de acceso: Usted debe disponer de la facilidad de dar una sola contraseña que funcione para todos los servidores y todos los servicios de la red.
- Transparencia de duplicación: A usted no debe interesarle cuantas copias existan de un recurso.
- Transparencia de acceso local/remoto: Usted debe estar en condiciones de trabajar con cualquier recurso de la red como si este se encontrara en la máquina local. El sistema operativo de red debe manejar los controles de acceso y proporcionar servicios de directorio.
- Transparencia de tiempo distribuido: Usted no debe percibir diferencias de horario entre servidores. El sistema operativo de red debe sincronizar los relojes de todos los servidores.
- Transparencia de fallas: Usted debe estar protegido contra fallas de la red. El sistema operativo de red debe manejar los reprocesamientos y

reconexiones de sesión. También debe ofrecer ciertos niveles de redundancia de servicios para la tolerancia de fallas.

- Transparencia de administración: Sólo se debe ver una interfaz de administración de sistema único. El sistema operativo de red debe estar integrado a los servicios de administración locales.

Mecanismos de Comunicación

Casi todas las primeras aplicaciones Cliente/Servidor fueron implementadas con protocolos de igual a igual para conversaciones como sockets y TLI de la suite de protocolos TCP/IP, NetBIOS y Named Pipes, entre otros. Entonces habían pocas opciones. Estos protocolos de bajo nivel son difíciles de codificar y mantener. Los programadores usan ahora RPC, MOM y ORB, que ofrecen mayores niveles de abstracción. Así, los protocolos para conversaciones sólo deben ser usados por aplicaciones del usuario o software de sistema muy exigente.

Llamada a Procedimientos Remotos (RPC)

Las RPC ocultan los detalles de la red mediante el empleo del mecanismo ordinario de llamada a procedimiento conocido por todos los programadores. Un

proceso cliente llama a una función en una máquina remota y se detiene hasta que recibe los resultados. Los parámetros se transmiten como en cualquier proceso ordinario. La RPC, al igual que un procedimiento ordinario, es sincrónica. El proceso que permite la llamada espera hasta que obtiene los resultados. En forma oculta, el software de RPC de tiempo de ejecución reúne valores para los parámetros, forma un mensaje y lo envía al computador remoto. El proceso en el computador remoto atiende la solicitud, desempaqueta los parámetros, llama al procedimiento y envía la respuesta al proceso cliente.

Manejo de Mensajes y Colas (MOM)

El middleware orientado a mensajes (MOM) es una pieza clave de middleware, absolutamente esencial para una clase de productos Cliente/Servidor. Si su aplicación puede tolerar cierto nivel de respuestas independientes del tiempo, MOM representa la vía más fácil para la creación de sistemas Cliente/Servidor empresariales. Contribuye también a la creación de sistemas Cliente/Servidor capaces de acumular en colas de espera transacciones en curso y ejecutar una carga en masa cuando es posible. El manejo de mensajes y la formación de colas de MOM permite la comunicación en red entre procesos clientes y procesos servidores sin que estén vinculados por una conexión lógica privada

especial. Clientes y Servidores pueden operar a diferentes tiempos. Todos se comunican insertando mensajes en colas y retirando mensajes de colas.

2.5 Autenticación

Autenticar es saber quién es. Además, demostrar que *"es quien dice ser"*. Es decir, se debe demostrar al sistema que en realidad es la persona autorizada para ingresar. El ejemplo práctico es el User y password:

User: cmartin

El sistema dice: Está bien, eres cmartin, demuéstralo.

A continuación pide una clave:

password: xxxxxx

Si la persona que desea ingresar al sistema es la autorizada, el User y password serán los correctos y podrá entrar al sistema; caso contrario, no lo hará.

2.6 Sistema de Administración de Base de Datos

El sistema de administración de base de datos (DBMS) que empleamos es SQL Server 7.0 de Microsoft, el cual forma la capa tres de nuestra arquitectura de tres capas. En el proceso servidor implementado, que se encuentra en la capa intermedia, está implementada una parte de nuestra lógica de aplicación. La otra parte de la lógica de aplicación se encuentra en la forma de stored procedures.
En los métodos de los objetos del servidor existen llamadas a estos stored procedures. La conexión con la base de datos es vía jdbc.
Creemos que es importante el uso de stored procedures ya que mejoran los tiempos de respuesta del sistema debido a que están formados por sentencia SQL precompiladas y almacenadas como parte de la base de datos.

Procesos Servidores de Transacciones

Un proceso servidor de transacciones es conocido en la terminología Cliente/Servidor como un TP Monitor. Existen a su vez dos tipos de TP Monitor: TP Heavy y TP Lite. Un TP Heavy es un proceso servidor dedicado exclusivamente a la administración de transacciones. Un TP Heavy en cambio es básicamente una base de datos que implementa stored procedures.

Una transacción es un bloque de código, el cual se ejecuta todo o simplemente no se ejecuta. La unidad de trabajo y administración para un TP Monitor es la transacción. Para un TP Heavy una transacción es equivalente a un stored procedure o a un conjunto de ellos. La transacción entonces, está formada por un conjunto de sentencias SQL.

Las sentencias SQL aciertan o fallan todas como una sola unidad para mantener el concepto de transacción. A estas instrucciones SQL agrupadas se les conoce como transacciones. A estas aplicaciones se les llama aplicaciones de transacciones en línea. Tienden a ser aplicaciones de misión crítica ya que se requieren tiempos de respuesta lo más pequeño posible.

Por todo lo expuesto estamos demostrando que el sistema desarrollado hace uso de un TP Lite, ya que es una aplicación de misión crítica y lo necesita para tener seguridad y un rendimiento.

3. PROCESO SERVIDOR

3.1 Esquema del Proceso Servidor

La aplicación es una solución Cliente/Servidor 3-tier. En esta sección, primero explicaremos los elementos que conforman cada una de las tres capas, luego mostraremos las interfaces IDL definidas que el proceso servidor expone a sus procesos clientes.

En nuestro esquema 3-tier, los procesos clientes manipulan objetos del proceso servidor. Por otro lado, los objetos del proceso servidor se conectan una o más veces vía JDBC con el DBMS.

La parte cliente del sistema está formada por dos tipos de clientes: MMVuelosClienteApplet que es un applet de Java y ClientVuelos que es una aplicación standalone.

La capa dos consiste de siete clases principales: 1) ManejaPoolServerObjectsVuelos que maneja un pool de objetos del proceso servidor, 2) Un pool de objetos de servidor de tipo MMVuelos y 3) Un pool de cinco objetos de tipo: Cliente, Reservación, Recorrido, Ubicación y Vuelo; que son objetos que trabajan directamente con la base de datos mediante conexiones JDBC.

La tercera capa la forma el proceso manejador de la base de datos, junto con parte de la lógica de aplicación en forma de stored procedures.

El objeto ManejaPoolServerObjectsVuelos mantiene un pool de objetos de servidor que asigna a los procesos clientes sobre la base "El primero que llega". Este objeto entrega objetos de tipo MMVuelos.

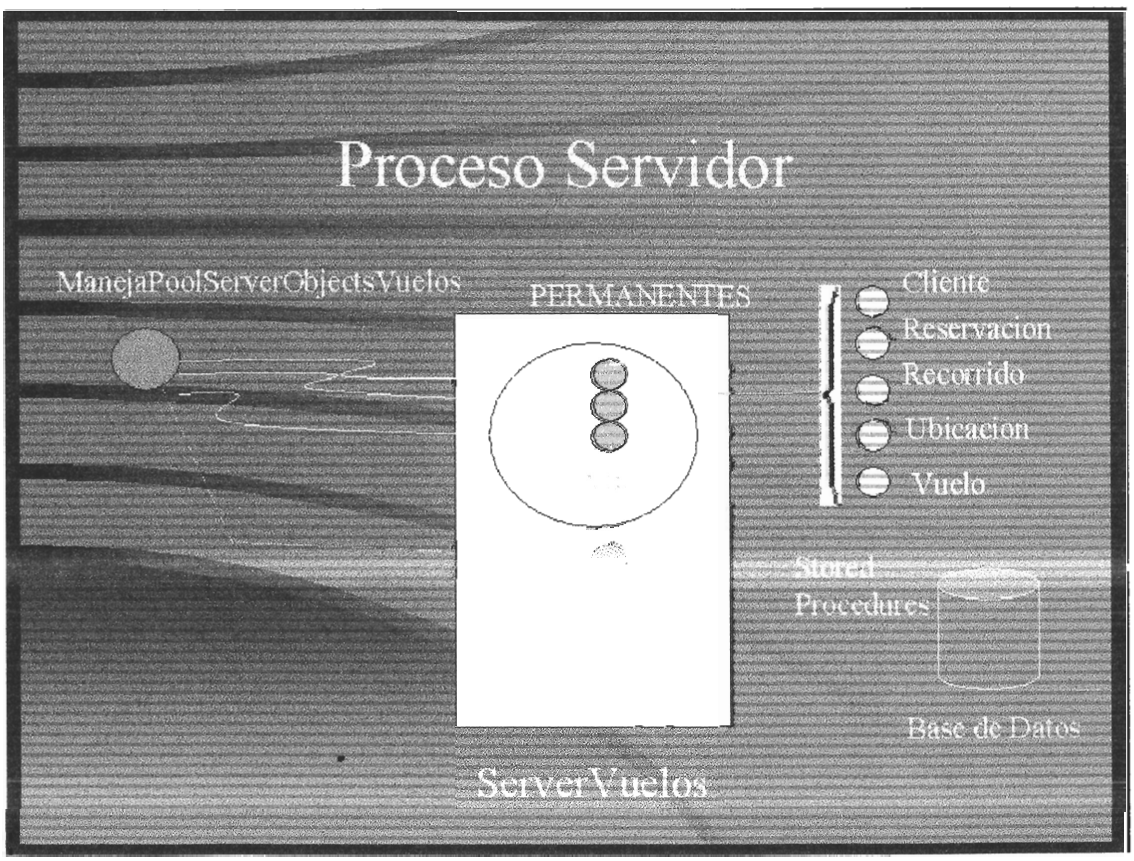


Figura 3.1 Esquema del Proceso Servidor

Cada objeto de tipo MMVuelos levanta sus propios objetos de tipo Cliente, Reservación, Recorrido, Ubicación y Vuelo.

La idea aquí es crear un planificador que maneje un pool de "prestarted multithreaded server objects". El planificador entrega éstos objetos por demanda. Un proceso cliente debe liberar el objeto de tipo de MMVuelos que le fue "entregado" una vez que éste complete su trabajo.

El Contrato IDL de MMVuelos

```
module Vuelos
{
  struct ubicacionConsulta
  {
    string id;
    string dsc;
  };
  typedef sequence<ubicacionConsulta> ubicacionSeq;
  struct recorridoConsulta
  {
    long id;
    string origen;
```

```
  string destino;
  long millas;
  long costo;
};
typedef sequence<recorridoConsulta> recorridoSeq;

struct vueloConsulta
{
  long id;
  long recorridoid;
  string partida;
  string llegada;
  long asientos;
  string cnl;
  string origen;
  string destino;
  long millas;
  long costo;
};
typedef sequence<vueloConsulta> vueloSeq;
```

```
struct reservacionConsulta
{
  long vueloid;
  long numreservas;
  long costototal;
  string fechareserva;
  string reservacnl;
};
typedef sequence<reservacionConsulta> reservacionSeq;

interface Cliente
{
  attribute long ClienteId;
  attribute string Apellidos;
  attribute string Nombres;
  attribute string Usuario;
  attribute string Clave;
  attribute string Email;
  attribute string Pais;
  attribute string Ciudad;
  attribute string Direccion;
```

```
 attribute long MillasTotal;

 boolean AsignaCliente(in string CliApellidos,
                       in string CliNombres,
                       in string CliUsuario,
                       in string CliClave,
                       in string CliEmail,
                       in string CliPais,
                       in string CliCiudad,
                       in string CliDireccion);

 boolean ValidarIngreso(in string CliUsuario,
                        in string CliClave);

};

interface Recorrido
{
  attribute long RecorridoId;
  attribute string Origen;
  attribute string Destino;
```

```
  attribute long Millas;
  attribute long Costo;

  boolean AsignaRecorrido(in string ReOrigen,
                          in string ReDestino,
                          in long ReMillas,
                          in long ReCosto);

  recorridoSeq ConsultaRecorridos();

};

interface Ubicacion
{
  attribute string UbicacionId;
  attribute string UbicacionDsc;

  boolean AsignaUbicacion(in string UbId,
                          in string UbDescripcion);

  ubicacionSeq ConsultaUbicaciones();
```

```
};

interface Vuelo
{
  attribute long VueloId;
  attribute long RecorridoId;
  attribute string FechaPartida;
  attribute string FechaLlegada;
  attribute long Asientos;
  attribute string VueloCnl;

  boolean AsignaVuelo(in long RecorridoIdVu,
                      in string FechaPartidaVu,
                      in string FechaLlegadaVu,
                      in long AsientosVu);

  vueloSeq ConsultaVuelos();

  boolean CancelaVuelo(in long VueloIdVu);
```

```
};
interface Reservacion
{
  attribute long ClienteId;
  attribute long VueloId;
  attribute long NumReservas;
  attribute long CostoTotal;
  attribute string FechaReserva;
  attribute string ReservaCnl;

  boolean AsignaReserva(in long ClienteIdRe,
                        in long VueloIdRe,
                        in long NumReservasRe);

  reservacionSeq ConsultaReservas(in long ClienteIdRe);

  boolean CancelaReserva(in long ClienteIdRe,
                         in long VueloIdRe,
                         in long NumReservasRe);

};
```

```
interface MMVuelos
{
  Cliente      SolicitudClienteObject();
  Recorrido    SolicitudRecorridoObject();
  Reservacion  SolicitudReservacionObject();
  Ubicacion    SolicitudUbicacionObject();
  Vuelo        SolicitudVueloObject();
};
interface ManejaPoolServerObjectsVuelos
{
  MMvuelos ReservarMMVuelosObject();
  boolean  LiberarMMVuelosObject(in MMvuelos MMVuelosObject);
};

};
```

3.2 Los Objetos del Proceso Servidor

Aquí están las clases Java que el lado servidor soporta:

- **ServerVuelos** provee el método *main* para el proceso servidor. Este cumple las siguientes funciones: 1) Inicializa el ORB, 2) Crea un objeto de tipo ManejaPoolServerObjectsVuelosImpl y le envía como parámetro el tamaño del pool de objetos de servidor-Este es un parámetro que se pasa a *main*, 3) Invoca *orb.connect* para registrar el objeto con el ORB, 4) Registra el recientemente creado objeto de tipo ManejaPoolServerObjectsVuelosImpl con el Servicio de Nombramiento y 5) Entra en un lazo infinito de espera para los requerimientos de los procesos clientes entrantes.
- **ManejaPoolServerObjectsVuelosImpl** implementa la interface ManejaPoolServerObjectsVuelos definida en el "contrato" IDL. Su método constructor crea un pool de objetos de tipo MMVuelos y almacena sus referencias en un arreglo de objetos de tipo StatusPoolServerObjectsVuelos. Finalmente invoca *orb.connect* para registrar cada uno de los recientemente creados objetos de tipo MMVuelos con el ORB. La clase implementa dos métodos definidos en la interface: *ReservarMMVuelosObject* y *LiberarMMVuelosObject*.

- **StatusPoolServerObjectsVuelos** implementa un estructura con dos campos: *referencia* y *enUso*. El primer campo almacena la referencia de un objeto de tipo MMVuelosImpl. El segundo campo contiene su estatus actual. El objeto de tipo ManejaPoolServerObjectsVuelosImpl mantiene un arreglo de este par de campos para poder tener control sobre el pool de objetos de servidor que maneja.

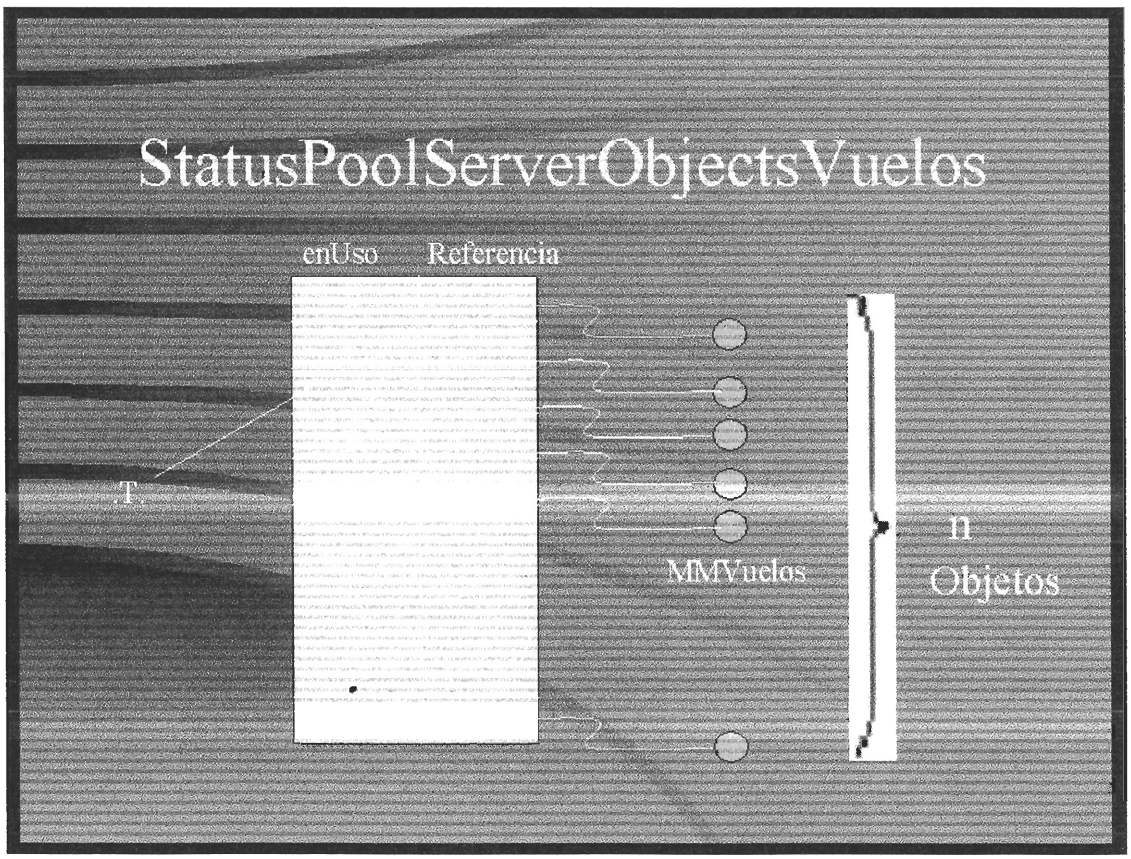


Figura 3.2 Implementación del Pool de Objetos

- **MMVuelosImpl** implementa la interface MMVuelos definida en el "contrato" IDL. La clase constructor crea cinco objetos de tipo ClienteImpl, ReservacionImpl, RecorridoImpl, UbicacionImpl y VueloImpl. Esta clase implementa los cinco métodos siguientes: SolicitudClienteObject, SolicitudReservacionObject, SolicitudRecorridoObject, SolicitudUbicacionObject y SolicitudVueloObject. Cada uno de éstos métodos permiten al proceso cliente "solicitar" cada uno de los cinco tipos de objetos mencionados.
- **ClienteImpl** implementa la interface Cliente. En sus propiedades contiene toda la información concerniente a los clientes. Adicionalmente contiene los métodos necesarios para ingresar nuevos clientes así como para poder validar su ingreso para el uso del sistema.
- **ReservacionImpl** implementa la interface Reservación. En sus propiedades contiene toda la información necesaria para realizar, consultar o cancelar reservaciones. Los métodos que posee permiten realizar todo lo mencionado.
- **RecorridoImpl** implementa la interface Recorrido. En sus propiedades contiene toda la información de un recorrido. Sus métodos nos permiten hacer consultas e ingresos de nuevos recorridos.

- **UbicacionImpl** implementa la interface Ubicación. En sus propiedades se puede almacenar el código de una ubicación así como su descripción. Los métodos nos permiten ingresar nuevas ubicaciones y consultarlas.
- **VueloImpl** implementa la interface Vuelo. En sus propiedades se almacena la información de los vuelos. Sus métodos permiten ingresar, cancelar y realizar consulta de vuelos.

Cada uno de estos cinco últimos tipos de objetos son los que se encargan de conectarse a la base de datos interactuando con el DBMS directamente.

4. BASE DE DATOS

4.1 Justificación de su Uso

Al ser diseñado el esquema del sistema de reservación de vuelos, surgió la necesidad de utilizar una herramienta de almacenamiento de datos.

La herramienta a seleccionar debía permitirnos implementar bases de datos relacionales, poseer un buen motor de base de datos y adicionalmente brindar el soporte para la creación de stored procedures. El producto que acogió nuestras expectativas fue SQL Server 7.0 de Microsoft.

Existen otros productos que también tienen las características que necesitamos, pero elegimos Microsoft porque tenemos mucha confianza en sus productos.

La base de datos del sistema de Reservaciones de Vuelos consta de cinco tablas que son:

- **Clientes**: Contiene los datos personales de los clientes de la aerolínea.
- **Reservaciones**: Almacena información de las reservaciones de vuelos de los clientes.
- **Recorridos**: Aquí se guarda las rutas y costos de los recorridos que seguirán los vuelos disponibles en la aerolínea.

- **Ubicaciones**: Contiene las ciudades que serán tomadas en cuenta como origen y destino de los vuelos.
- **Vuelos**: Guarda toda información que se necesita de los vuelos.

4.2 Diagrama Entidad Relación

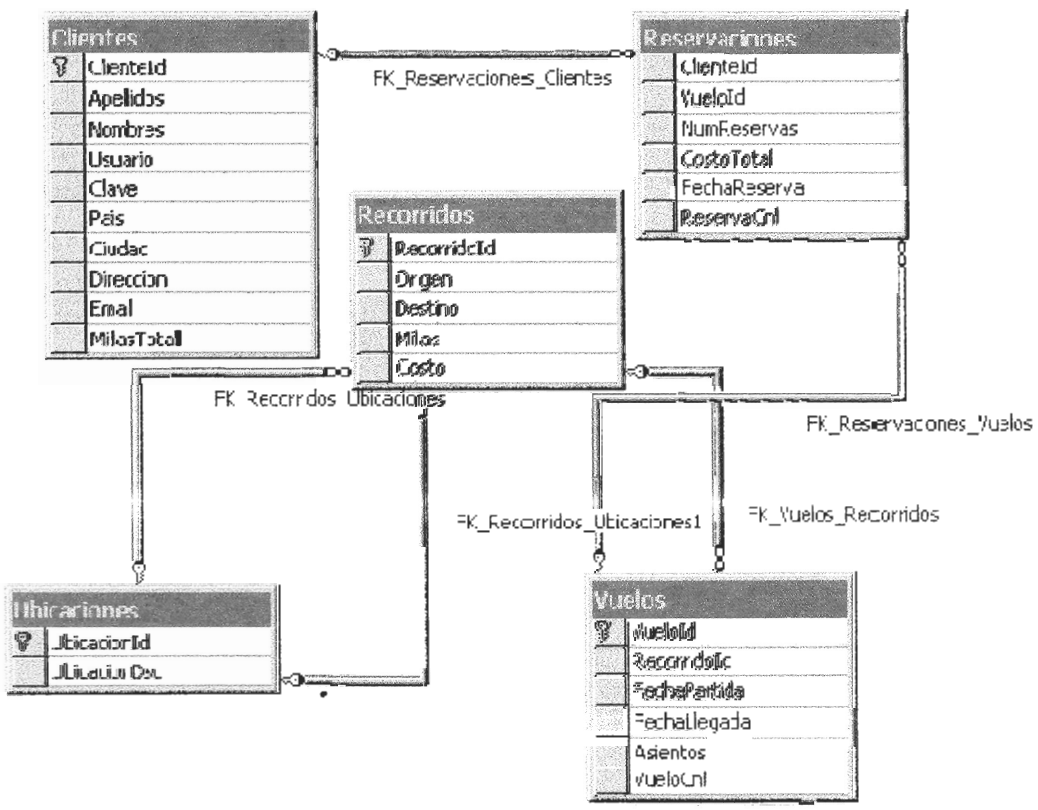


figura 4.1 Diagrama Entidad Relación

4.3 Tabla Clientes

Clientes

Column Name	Datatype	Length	Precision	Scale	Allow Nulls
ClienteId	int	4	10	0	☐
Apellidos	varchar	50	0	0	☑
Nombres	varchar	50	0	0	☑
Usuario	varchar	15	0	0	☑
Clave	varchar	10	0	0	☑
Pais	varchar	50	0	0	☑
Ciudad	varchar	50	0	0	☑
Direccion	varchar	70	0	0	☑
Email	varchar	30	0	0	☑
MillasTotal	numeric	9	18	0	☑
					☐

4.4 Tabla Reservaciones

Reservaciones

Column Name	Datatype	Length	Precision	Scale	Allow Nulls
ClienteId	int	4	10	0	☐
VueloId	int	4	10	0	☐
NumReservas	int	4	10	0	☑
CostoTotal	int	4	10	0	☑
FechaReserva	datetime	8	0	0	☑
ReservaCnl	char	1	0	0	☑
					☐

4.5 Tabla Vuelos

Vuelos

Column Name	Datatype	Length	Precision	Scale	Allow Nulls
VueloId	int	4	10	0	☐
RecorridoId	int	4	10	0	☑
FechaPartida	datetime	8	0	0	☑
FechaLlegada	datetime	8	0	0	☑
Asientos	int	4	10	0	☑
VueloCnl	char	1	0	0	☑
					☐

4.6 Tabla Recorridos

Recorridos

Column Name	Datatype	Length	Precision	Scale	Allow Nulls
RecorridoId	int	4	10	0	☐
Origen	char	2	0	0	☑
Destino	char	2	0	0	☑
Millas	int	4	10	0	☑
Costo	int	4	10	0	☑
					☐

4.7 Tabla Ubicaciones

Ubicaciones

Column Name	Datatype	Length	Precision	Scale	Allow Nulls
UbicacionId	char	2	0	0	☐
UbicacionDsc	varchar	50	0	0	☑

5. CONCLUSIONES

- Para toda la industria de la computación, con la notable excepción de Microsoft Corporation, la siguiente generación de middleware es CORBA.

- El desarrollo de aplicaciones distribuidas con esquemas 3-tier, n-tier se facilita cuando empleamos una tecnología que aporte con un "Bus de Objetos" y que aplique los conceptos de ORB.

- La computación de Internet no reemplaza a la arquitectura Cliente/Servidor, ya que ésta es en si misma Cliente/servidor.

- Los DBMS Cliente/Servidor tienen 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.

- La siguiente generación de sistemas distribuidos estarán basados en objetos distribuidos o componentes.

- En un contexto distribuido es donde creemos que los objetos brindarán todo su potencial.

BIBLIOGRAFIA

1. ARNOLD Y GOSLING, El Lenguaje de Programación Java, Addison-Wesley/Domo, 1997.

2. MICROSOFT CORPORATION, Mastering Distributed Application Design, Student Workbook, Microsoft, 1998.

3. ORFALI, HARKEY Y EDWARDS, The Essential Client/Server Survival Guide, WILEY, Second Edition, 1998.

4. ORFALI Y HARKEY, Cliente/Server Programming With JAVA and CORBA, WILEY, Second Edition, 1998.

5. WEHLING Y BHARAT, Aproveche las noches con Java, Prentice-Hall Hispanoamericana, 1997.