ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"APLICACIÓN DE LA TECNOLOGÍA CLIENTE/SERVIDOR EN TRES CAPAS CON OBJETOS DISTRIBUIDOS EN LA RESERVACIÓN DE HABITACIONES DE UN HOTEL"

TESIS DE GRADO

Previa a la obtención del Título de:

INGENIERO EN COMPUTACIÓN
ESPECIALIZACIÓN EN SISTEMAS TECNOLÓGICOS

Presentada por:

JUAN CARLOS CRUZ RODRÍGUEZ
ITALO BOLÍVAR GALARZA ESPINOZA

GUAYAQUIL – ECUADOR

AÑO

2005

AGRADECIMIENTO

ING. FABRICIO ECHEVERRÍA Director de Tesis, por su colaboración, impulso y todas las facilidades brindadas para la elaboración y sustento de esta tesis.

DEDICATORIA

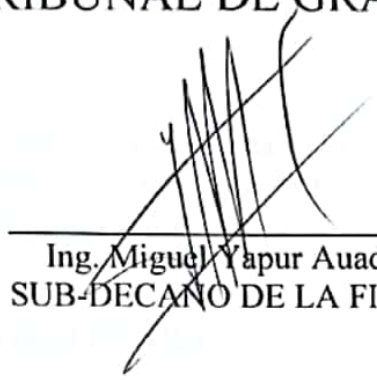
A NUESTROS PADRES

A NUESTROS HERMANOS

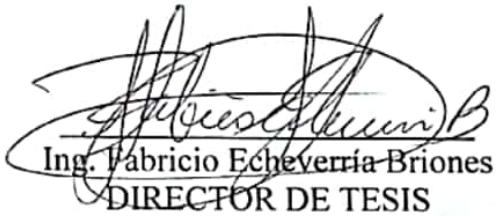
A MI HIJO

A MI ÑAÑA CLARITA

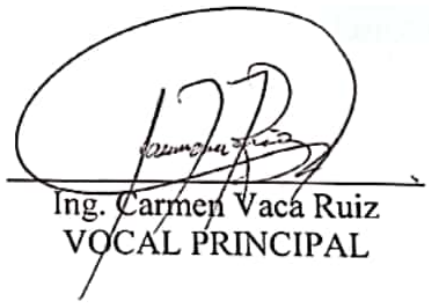
TRIBUNAL DE GRADO



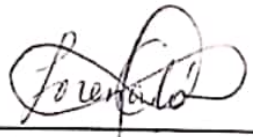
Ing. Miguel Yapur Auad
SUB-DECANO DE LA FIEC



Ing. Fabricio Echeverría Briones
DIRECTOR DE TESIS



Ing. Carmen Vaca Ruiz
VOCAL PRINCIPAL

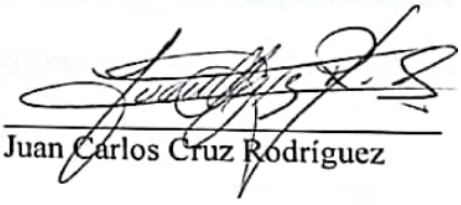


Ing. Lorena Carló Unda
VOCAL PRINCIPAL

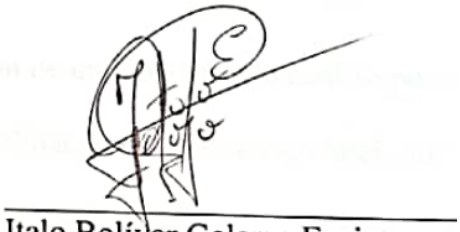
DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta Tesis de Grado, me corresponde exclusivamente; y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL"

(Reglamento de graduación de la ESPOL).



Juan Carlos Cruz Rodríguez



Italo Bolívar Galarza Espinoza

RESUMEN

Las aplicaciones de múltiples-capas, requieren comprender y conocer la forma de trabajar con objetos distribuidos, sobre todo para maximizar la utilidad y reducir la complejidad.

En un sistema distribuido la ubicación de los datos o de las aplicaciones no es importante, ya que es totalmente transparente para el cliente. Se requiere de una implementación completa del Directorio de Servicios, con esto se logra que la aplicación esté aislada de los procesos y los datos del sistema.

Se implementó una aplicación para la administración de un Hotel con la cual se pueda realizar las siguientes transacciones: reservas de habitaciones, check-in, check-out y emisión de facturas. Esta aplicación utiliza una arquitectura Cliente/Servidor en múltiples capas. Para la implementación se utilizaron tecnologías de última generación como son: CORBA, JDBC y Applets, aplicando conceptos para la obtención de recursos distribuidos desde el Internet. Para los clientes se han implementado Applets que invocan a objetos en el servidor (objetos CORBA), a través del protocolo IIOP. Toda la lógica del negocio se encuentra en los objetos servidores y estos almacenan los datos en la base de datos SQL con soporte JDBC.

ÍNDICE GENERAL

ÍNDICE DE GRÁFICOS

INTRODUCCIÓN

Antecedentes

La tecnología Cliente/Servidor apareció como respuesta a las necesidades de desarrollar sistemas distribuidos. Esta tecnología tiene varios componentes que distribuyen los elementos básicos de un sistema: Front-End, Middleware, Back-End. A través del tiempo esta tecnología ha ido evolucionando desde esquemas de Dos-Capas hasta esquemas de Múltiples-Capas.

Dentro del esquema Múltiples-Capas la organización OMG (Object Management Group) desarrolló la tecnología CORBA, que en la actualidad tiene mucha acogida a nivel Europeo y se está empezando a utilizar en el mercado norteamericano. Esta popularidad se debe a que CORBA se convirtió en una de las impulsadoras de la comunicación entre servicios de programas para el desarrollo de sistemas distribuidos. Las principales

funciones de un sistema distribuido son responder a necesidades de interoperabilidad y sobre todo permitir que las aplicaciones se comuniquen entre ellas sin importar su ubicación física, CORBA permitió que se pueda emplear metodologías de OOD (Diseño Orientado a Objetos) provocando que se cambien las disciplinas de análisis, desarrollo, e implementación, brindándonos la flexibilidad para reutilización, facilidad para el mantenimiento y utilización de integración con otros componentes internos o externos.

Propósito

El propósito de esta tesis es aplicar la tecnología CORBA y demostrar su funcionalidad por medio del desarrollo e implantación de una aplicación real. Además se muestra la implementación de las tres capas esenciales: Capa de interface grafica, capa de lógica del negocio y la capa de datos. Para el desarrollo del proyecto se usa la metodología de Análisis, Diseño e Implementación Orientada a Objetos.

Proceso y Herramientas para el desarrollo del Proyecto

Se implementó una aplicación para la administración de un Hotel, con la cual se pueda realizar las siguientes transacciones: reservas de habitaciones, check-in, check-out y emisión de factura, está aplicación utiliza una arquitectura Cliente/Servidor en múltiples capas, para demostrar la funcionalidad de CORBA, JDBC y Applets, y la obtención de recursos distribuidos desde Internet.

Para el desarrollo del proyecto, revisamos páginas web de varios hoteles de la ciudad de Guayaquil como son: Hotel Oro Verde, Hotel Colon, Sheraton, y el Hotel Royal de Colombia, observamos los requerimientos básicos y comunes de cada uno de ellos además de otras funcionalidades necesarias en un sistema de administración de hoteles.

Como siguiente paso se hizo el análisis y diseño orientado a objetos utilizando OMT (Tecnología de Modelamiento Orientado a Objetos / Rumbaugh), aplicando los conocimientos de los integrantes del grupo adquiridos en la materia Análisis y Diseño Orientado a Objetos.

Como último paso el proyecto fue implementado haciendo uso de las siguientes herramientas: Eclipse, SQL Server 2000, Windows Server 2000, Windows XP, VisiBroker for Java, Java 2 SDK y Apache Web Server.

Vista General del documento

En los capítulos 1 y 2 se describe todo el proyecto y proceso utilizado para la implementación e implantación de la aplicación. Se comienza describiendo toda la tecnología nueva que se utiliza. Luego se describen las partes esenciales del proyecto para cada una de las tres capas: Capa de Interface (CLIENTE), Capa del Negocio (SERVIDOR) y Capa de Datos (BASE DE DATOS).

En los capítulos 3 y 4 se muestra el proceso para el desarrollo del proyecto, esto es análisis y diseño de la parte lógica, análisis y diseño de la base de datos. Con el análisis y diseño listo se procede a implementar él código de la aplicación, para esto se utiliza JAVA y los procedimientos almacenados en SQL SERVER. Al final del capítulo 4 se encuentra la sección de implantación del proyecto.

En los capítulos finales tenemos nuestras conclusiones, el glosario de términos y los anexos donde indicamos los documentos adicionales para una mejor explicación del proyecto.

CAPÍTULO 1

1 REFERENCIA TECNOLÓGICA

1.1 Definición de la tecnología Cliente/Servidor

Cliente/Servidor es una arquitectura para el desarrollo que divide a una aplicación en varios componentes. En su forma básica deben existir por lo menos dos componentes: cliente y servidor. El proceso servidor puede ser ejecutado en las diversas plataformas existentes en el mercado, los procesos clientes se comunican a través de la red usando uno o varios protocolos de LAN o WAN para interactuar con el proceso servidor.

Uno de los objetivos de la tecnología mencionada, es lograr la comunicación entre hosts (clientes y servidor) de manera transparente, inclusive entre hosts de diferentes arquitecturas y/o plataformas.

1.2 Esquema Cliente/Servidor Básico

La arquitectura Cliente/Servidor ha ido evolucionado como primera respuesta a las mejoras de rendimiento de las PC y las redes Ethernet. El segundo cambio y/o revolución importante se observó en las WAN. En la actualidad el mejor esquema Cliente/Servidor y de gran difusión es el Web, que es un esquema básico de dos capas.

Un cliente es definido como alguien que requiere servicios, y un servidor es definido como alguien que provee esos servicios. Un host puede ser ambos cliente y servidor, depende de la configuración de los servicios. Los hosts clientes acceden a los recursos y aplicaciones que residen en el host servidor. La base de datos entrega registros y/o información a los hosts clientes usando componentes en ambos lados, es decir, en el cliente y en el servidor.

El desarrollo de este tipo de aplicaciones requiere procesamiento transaccional, conocimiento de comunicaciones, interfaces de usuario, diseño de bases de datos, objetos distribuidos e Internet. Dependiendo de la aplicación a desarrollar se hará uso de parte o de todo lo anteriormente mencionado.

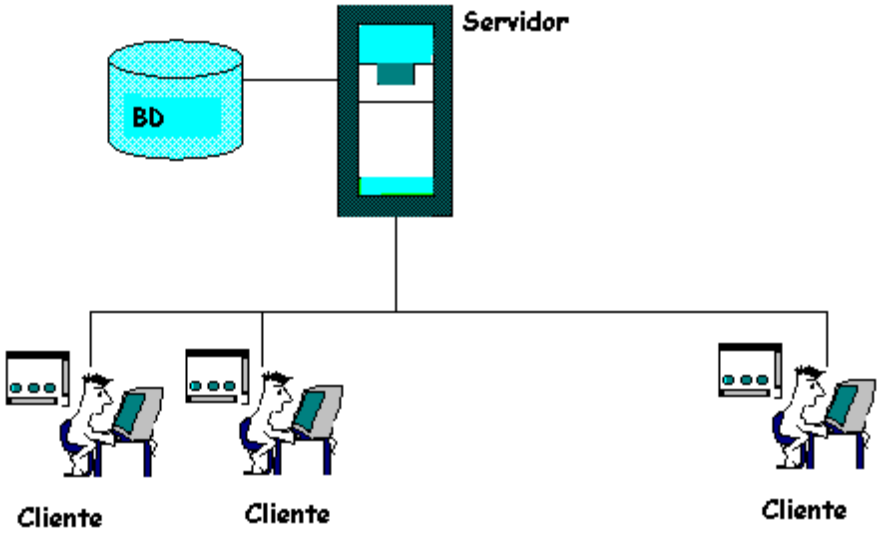


Figura 1. Esquema Básico Cliente/Servidor

1.2.1 Características deseables del esquema Cliente/Servidor

Transparencia de ubicación.- El servidor es un proceso que puede residir en la misma máquina del cliente o en una máquina diferente que pertenezca a la red. El software

Cliente/Servidor usualmente oculta la ubicación del servidor a los clientes pero direccionando las llamadas a los servicios si es necesario. Un programa puede ser cliente, servidor o ambos.

Transparencia de Plataforma.- El software Cliente Servidor ideal es independiente del Hardware o de la plataforma donde se ejecuta (Sistema Operativo). El software tiene que ser capaz de trabajar entre plataformas heterogéneas.

Escalabilidad.- Los sistemas cliente servidor pueden ser escalados Horizontalmente o Verticalmente. El escalamiento horizontal consiste en agregar o quitar estaciones cliente, provocando un impacto en el desempeño. El escalamiento vertical se trata de migrar a máquinas servidores más rápidos y robustos.

1.2.2 Limitaciones del esquema Cliente/Servidor básico

Dentro del esquema Cliente/Servidor básico se tienen algunas "pequeñas" limitaciones que van depender de la cantidad de requerimientos clientes como son:

- Los clientes deben conocer previamente la dirección del servidor al cual se solicita el o los requerimientos.

- Para conocer dicha dirección se envía un mensaje de broadcast, lo cual genera tráfico en la red.
- Como sólo existe un servidor atendiendo a todos los clientes, se crea un "cuello de botella" en la interfaz de red, con lo que aumenta el tiempo de respuesta a los requerimientos.

1.3 Esquema Cliente/Servidor de Procesamiento Distribuido

El cambio en la velocidad de acceso de las redes, la integración de diferentes protocolos, y el poder de la comunicación entre diferentes ubicaciones, ha provocado la evolución de lo que conocemos como procesamiento distribuido Cliente/Servidor. Este tipo de esquema resultó de la combinación de lo siguiente:

Modelo Cliente/Servidor puro.- Los terminales clientes accesan a los recursos y aplicaciones del host servidor.

Modelo Peer-Processing.- Cualquier host en la red puede tener el rol de cliente o servidor. En ocasiones puede inclusive tener ambos roles, siendo cliente de un servicio y servidor de otro.

Modelo 2-Capas.- Con este esquema, la interface de usuario está ubicada en los desktops (equipos del lado del cliente), y la administración de la base de datos y servicios están del lado del servidor. La administración de los procesos está dividida entre las interfases de usuario y la administración de la base de datos. El servidor de base de datos provee procedimientos almacenados y triggers.

La arquitectura cliente/servidor de 2-Capas, es una buena solución para ambientes distribuidos cuando los grupos de trabajo comprenden alrededor de 100 personas trabajando en una LAN de manera simultánea. Cuando el número de usuarios es mayor a 100, el desempeño empieza a deteriorarse. Otra limitación de esta arquitectura es que la declaración de procedimientos almacenados es dependiente del vendedor del motor de base de datos DBMS (System Manager Data Base).

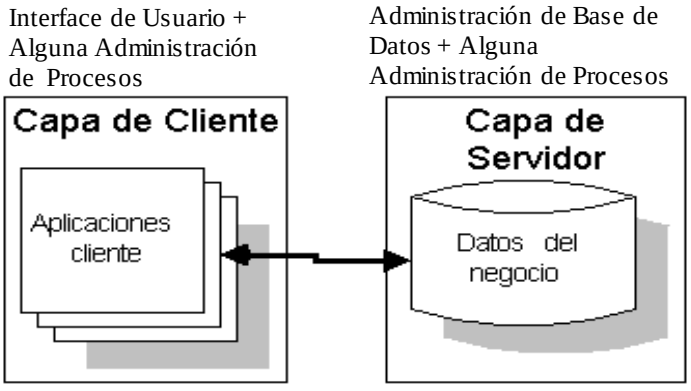


Figura 2. Diseño de la Arquitectura de Dos-Capas Cliente/Servidor.

Modelo 3-Capas.- Esta arquitectura sale a relucir en los años noventa, para cubrir las limitaciones de la arquitectura dos-Capas. La tercera capa (capa servidora intermedia) se ubica entre la interface de usuario (cliente) y los componentes de administración de datos (servidor). Esta capa intermedia provee procesos de administración donde se ejecutan las reglas y la lógica del negocio a las que tienen acceso cientos de usuarios.

Este esquema es el que se encarga de balancear la carga de trabajo entre los nodos y monitorear el estado de los procesos servidores. Los desarrolladores no tienen que preocuparse de fallas en el nodo, el acceso de los clientes (concurrencia), y la sincronización de recursos entre los múltiples nodos. Todo lo mencionado ocurre de manera transparente.

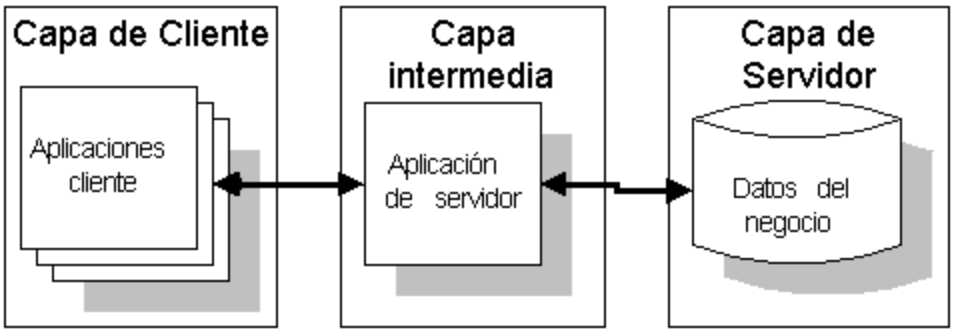


Figura 3. Tres Capas, distribución de la arquitectura Cliente/Servidor.

La capa intermedia, provee mejor desempeño, flexibilidad, mantenimiento, reusabilidad, y escalabilidad.

Para desarrollar una aplicación múltiples capas, es necesario conocer y entender la forma de trabajar con objetos distribuidos, sobre todo para maximizar la utilidad y disminuir la complejidad. Como se ha mencionado anteriormente en este documento, la ubicación física de los procesos servidores y los datos no es importante, se requiere de una implementación completa del Directorio de Servicios (localizar la dirección correcta, identificar usuarios y recursos, autenticación y seguridad). El directorio de servicios evita la necesidad de que la aplicación conozca donde se encuentran los procesos y los datos.

1.4 Middleware

El término Middleware identifica a un programa que se ejecuta como un proceso que cubre lo necesario para soportar la interacción entre las capas de la aplicación cliente y la aplicación servidor. Middleware es la interconexión que permite a un cliente obtener un servicio desde un servidor.

¿Dónde inicia y dónde termina el Middleware?. Comienza con un conjunto de API's (Application Programming Interfaces) sobre el lado del cliente que son usados para invocar un servicio, cubre la transmisión de los requerimientos sobre la RED y los resultados de respuesta. No incluye las interfaces de usuarios o la lógica de la aplicación.

Application Programming Interfaces (API), es una tecnología antigua que facilita el intercambio de mensajes o datos entre dos o más aplicaciones de software diferentes. El API es la interfaz virtual entre dos funciones de interconexión, como un procesador de texto y una hoja de cálculo. Esta tecnología se ha expandido desde simples llamadas a subrutinas, para incluir características que proveen interoperabilidad y la facilidad para la incorporación de cambios, y brindan apoyo para compartir datos entre múltiples aplicaciones. Un API es el software que se usa para apoyar la integración a nivel del sistema de múltiples aplicaciones-comerciales, productos de software o nuevos-desarrollos, dentro de aplicaciones existentes o nuevas. Los APIs son también un tipo de Middleware que comparten datos a través de diferentes plataformas. Esta es una característica importante cuando se hace nuevo desarrollo o se actualizan los sistemas distribuidos existentes.

Hay cuatro tipos de APIs que se utilizan para compartir datos entre diferentes aplicaciones de software diferentes ya sea un hosts o plataformas distribuidas:

- Remote Procedure Calls (RPC).
- Standard Query Languague (SQL).
- File transfer.
- Message delivery.

Usando RPC.- Los programas se comunican vía procedimientos (tareas), que actúan sobre buffers de datos compartidos.

SQL.- Es un lenguaje de acceso a datos que permiten compartir datos entre aplicaciones pero accesando a una base de datos común.

File transfer.- Permite compartir datos enviando archivos estructurados entre aplicaciones.

Message delivery.- Las aplicaciones comparten datos por una comunicación directa entre programas vía pequeños mensajes estructurados.

Como se muestra en la figura los servicios del middleware, son un conjunto de software distribuido, necesario para el soporte de interacciones entre Clientes y Servidores. Es el enlace que permite que un cliente obtenga un servicio de un servidor.

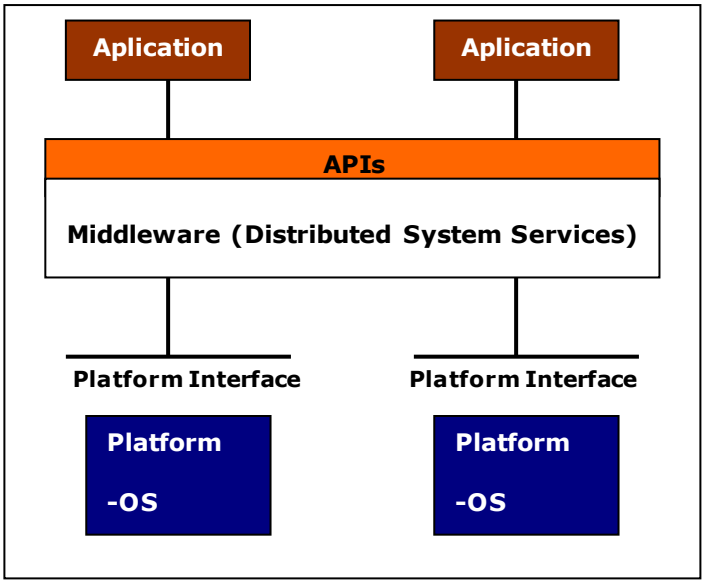


Figura 4. Uso de Middleware.

Existen diversas formas de Middleware. Entre las que tenemos:

Mecanismos de comunicación.- Son los protocolos al nivel de las capas de sesión, transporte y red, que permiten establecer sesiones entre clientes y servidores, por ejemplo: TCP/IP (Sockets), NetBEUI/NetBIOS.

Funciones especiales de los NOS.- Network Operating System (NOS), son funciones y procesos de apoyo que extienden la capacidad de los sistemas operativos con: funciones

de seguridad como niveles C2, funciones de administración, de archivos distribuidos como X.500, servicios distribuidos de tiempo, funciones para la invocación a procedimientos remotos y procesos servidores.

1.5 Tecnología CORBA

CORBA (Common Object Request Broker) es una tecnología que se incorpora dentro del middleware en la industria de las aplicaciones Cliente/Servidor Distribuidas, y es el producto de un consorcio llamado Object Management Group (OMG). Este consorcio incluye alrededor de 800 compañías de la industria de la computación, y es el principal competidor de Microsoft DCOM (Distributed Component Object Model). CORBA fue creado con el propósito de fomentar el uso de la tecnología orientada a objetos en sistemas distribuidos, además permite el diseño de componentes inteligentes para descubrirse entre ellos e ínteroperar sobre un bus de mensajes de objetos.

El secreto de la OMG es crear especificaciones de interfaces y no código. Las especificaciones son escritas en una interface neutral llamada Interface Definition Lenguaje (IDL). CORBA usa IDL's, que se pueden usar para definir APIs que controlen

problemas importantes como manejo de errores. CORBA brinda apoyo a requerimientos clientes que hacen peticiones de objetos distribuidos. Estos objetos permiten la interacción Cliente/Servidor dentro de la implementación específica del ORB (Object Request Broker), sin importar el lenguaje de programación en el que esté escrito el cliente o el servidor, lo que permite construir aplicaciones portables sin importar los lenguajes de programación, herramientas, y sistemas operativos.

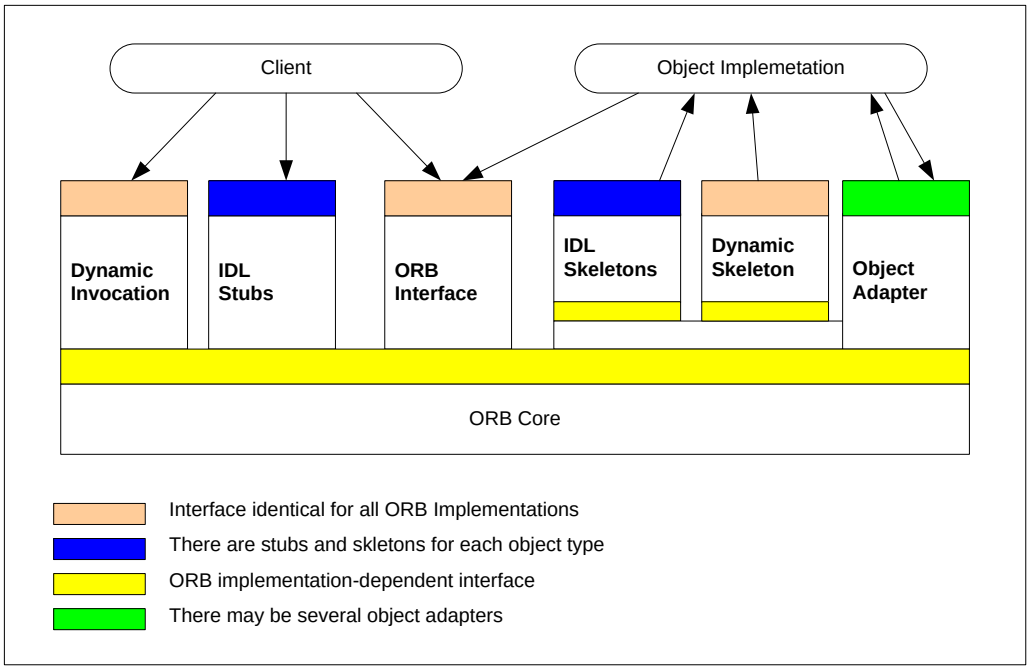


Figura 5. Estructura de Interfaces CORBA.

La OMG, sólo define especificaciones para aplicaciones y no lineamientos para un lenguaje de desarrollo específico. Esto lo realizan a través de la Object Management Architecture Guide (OMA Guide), que se define en cuatro elementos principales:

- **Object Request Broker ORB**, es una infraestructura de comunicación, a través de la cual las aplicaciones acceden a los servicios: CORBAServices, CORBAFacilities, ApplicationObjects, y a través del cual los objetos interactúan unos con otros. Define los objetos CORBA en un bus.
- **CORBAServices**, son considerados fundamentales para el desarrollo de sistemas distribuidos no-triviales. Estos servicios actualmente incluyen: Administración de eventos asíncronos, transacciones, persistencia, guardar el estado de un objeto en un stream de datos (externalization), concurrencia, habilidad para encontrar el nombre de un objeto (naming), relación de objetos y ciclo de vida. Define los objetos en el framework que se extienden en el bus.
- **CORBAFacilities**, puede ser útil en aplicaciones distribuidas en algunos escenarios, pero no es considerado universalmente aplicable como CORBAServices. CORBAFacilities incluye: la interfaz de usuario, información de administración, administración de tareas, y una variedad de framework horizontal y vertical que son usados directamente por los objetos del negocio.
- **Application Objects**, provee servicios que son particulares a una aplicación, o clases de una aplicación. Estos no son actualmente un tópico de estandarización

dentro de OMA pero son incluidos en el modelo referencial de OMA, Por ejemplo: Objetos, aplicaciones-específicas, apoyo a servicios básicos.

Los puntos mencionados anteriormente son los principios básicos de CORBA.

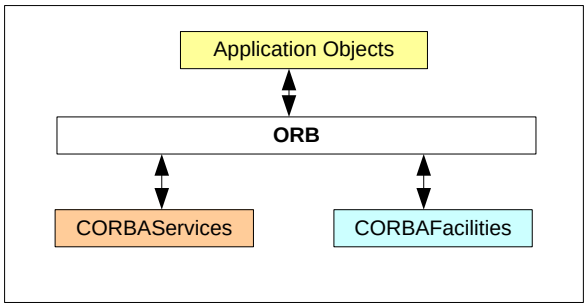


Figura 6. OMA (Object Management Architecture)

De los componentes mencionados, el **Object Request Broker** (ORB) es el encargado de interceptar las llamadas para encontrar un objeto que pueda implementar el requerimiento. El cliente no se preocupa de la ubicación del objeto, en qué lenguaje fue desarrollado o el sistema operativo donde reside y es ejecutado. El object request broker (ORB) es una tecnología middleware que administra la comunicación y el intercambio de datos entre objetos.

Los ORBs promueven la interoperabilidad de sistemas de objetos distribuidos, porque habilitan a los usuarios a construir los sistemas por partes y juntar luego los objetos de diferentes distribuidores, estos se comunican unos con otros vía el ORB. Los detalles de

la implementación de los ORBs, generalmente no son importantes para los desarrolladores que construyen sistemas distribuidos. Los desarrolladores sólo se preocupan por los detalles de interfaz de objeto. Esta forma de encapsulación de la información, refuerza el mantenimiento del sistema, ya que los detalles de comunicación de los objetos están ocultos a los desarrolladores y aislados de los ORBs.

El objetivo de la tecnología ORB es promover la comunicación de objetos a través de máquinas y software. Las funciones de la tecnología ORB son:

- La definición de la interfaz.
- Localización y la posible activación de objetos remotos.
- La comunicación entre clientes y objeto.

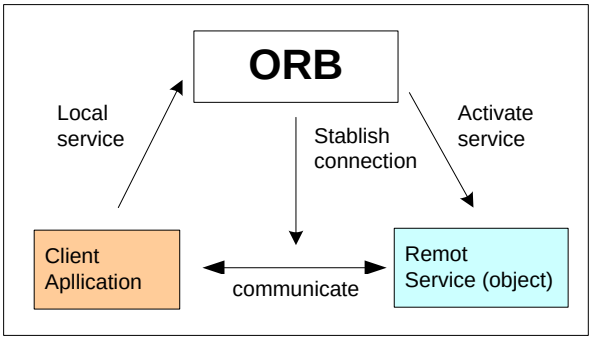


Figura 7. Object Request Broker

El componente **CORBAServices** es una colección de servicios agrupados dentro de un IDL. Encargado de crear componentes, y controlar el acceso a estos componentes, la OMG ha publicado 15 estándares de servicios de objetos entre los cuales tenemos:

- Persistent Service, provee una sola interface para almacenar componentes.
- Naming Service, permite localizar componentes usando su nombre.
- Transaction Service, provee una sincronización de transacciones a través de los elementos distribuidos de una aplicación cliente servidor.
- Trader Service, consiste en una publicación como la guía telefónica (páginas amarillas) para los objetos. Permite publicar sus servicios de acuerdo a lo que realizan (propiedades).

Los objetos CORBA se diferencian de los objetos típicos en las siguientes formas:

- Pueden estar ubicados en cualquier parte de la red.
- Pueden interoperar con objetos de otras plataformas.
- Pueden escribirse en cualquier lenguaje de programación.

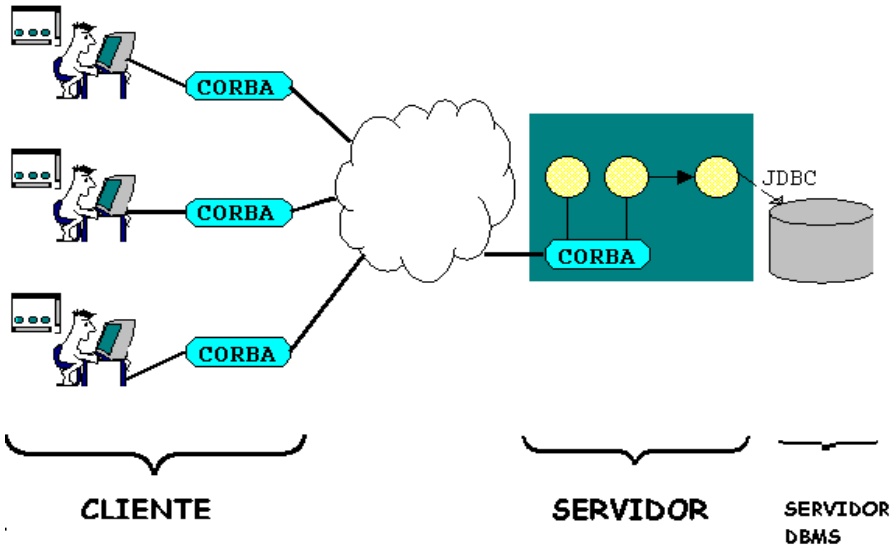


Figura 8. Modelamiento de Tres-Capas con CORBA

1.5.1 GIOP (General Inter.-ORB Protocol)

El GIOP especifica una sintaxis de transferencia estándar y un conjunto de formatos de mensaje para comunicación entre ORBs. El GIOP está construido específicamente para interacciones ORB-ORB, y está diseñado para trabajar directamente sobre cualquier protocolo de transporte orientado a conexión que cumpla con un conjunto mínimo de requerimientos. El protocolo es simple, escalable y relativamente fácil de implementar.

Si bien versiones del GIOP corriendo en diferentes capas de transporte no serían directamente compatibles, su similitud permitiría un enlace fácil y eficiente.

1.5.2 IIOP (Internet InterORB Protocol)

El IIOP especifica cómo los mensajes GIOP se intercambian utilizando conexiones TCP/IP. El IIOP especifica un protocolo de interoperabilidad estandarizado para Internet, proveyendo una correlación directa con otros ORBs compatibles basados en la capa de transporte más popular. Puede también ser utilizado como el protocolo entre "medios de enlaces".

El protocolo es apropiado para ser usado por cualquier ORB usando diferentes protocolos de Internet a menos que se necesite un protocolo alternativo en particular. En ese sentido representa al protocolo básico interORB para ambientes TCP/IP, una capa de transporte ampliamente utilizada.

La relación del IIOP con el GIOP es similar a la de un mapeo específico de lenguaje con IDL; el GIOP puede ser mapeado a un número de diferentes transportes y especifica los elementos de protocolo comunes a todos estos mapeos. El GIOP por sí mismo, sin embargo, no provee interoperabilidad completa, tal como IDL no puede ser utilizado para construir programas completos. El IIOP y otros mapeos similares a diferentes transportes son realizaciones concretas de las definiciones abstractas GIOP.

1.5.3 Protocolos InterORB específicos de ambiente (ESIOPs)

Esta especificación también provee un conjunto abierto de ESIOPs. Estos protocolos serían utilizados para ínteroperación en sitios de usuario donde se utiliza una infraestructura particular de redes o computación distribuida.

Debido a la oportunidad de desarrollo provista por el ambiente específico, los ESIOPs podrían soportar capacidades especializadas tales como aquellas relacionadas a seguridad y administración.

Si bien los ESIOPs pueden ser optimizados para ambientes particulares, se espera que todas las especificaciones de ESIOP cumplan las convenciones de interoperabilidad general de ORBs para permitir puenteo de manera simple. El soporte de puenteo interORB permite que se construyan puentes entre dominios ORB que utilizan IIOP y dominios ORB que utilizan un ESIOP particular.

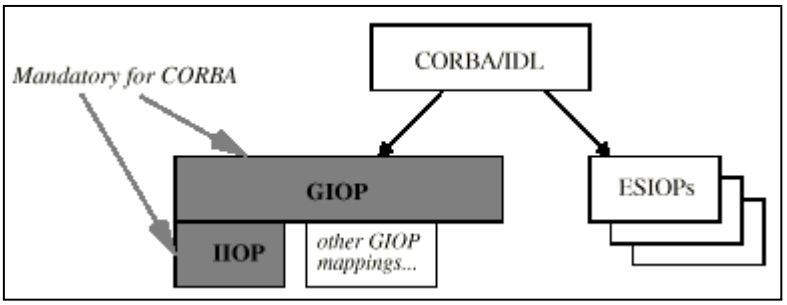


Figura 9. Capa elementos de Interoperabilidad

1.6 EJB (Enterprise Java Beans)

EJB es una especificación para sistemas distribuidos basada en componentes, usando tecnología Java. EJB describe:

- *Roles* y responsabilidades para desarrollo de software basado en componentes del lado del servidor.
- La *arquitectura* incluye servidores EJB, Contenedores y Beans.
- El conjunto de *servicios* incluye naming, transacciones y seguridad.
- *Interoperabilidad* con servidores de bases de datos, y aplicaciones CORBA.

La especificación de EJB tiene como objetivos: el desarrollo de software basado en componentes, separar la lógica del negocio del código del sistema, direccionar la aplicación hacia el ciclo de vida de la misma, ser compatible con CORBA (aplicaciones No-Java).

Arquitectura de EJB

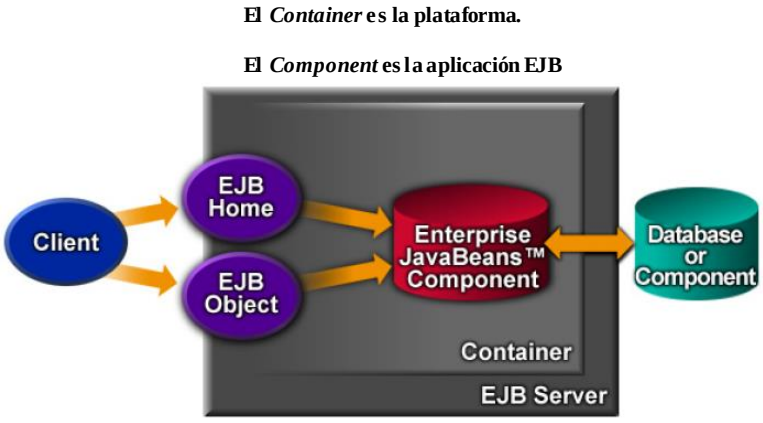


Figura 10. Arquitectura EJB.

La Plataforma EJB provee naming service usando JNDI, provee servicios transaccionales compatibles con OMG/OTS, administra el ciclo de vida del EJ Bean, hace disponible las interfaces EJ Bean con JNDI, provee servicios básicos de seguridad, administra la persistencia con DBMS y otros.

El EJ Application consiste de múltiples beans. Los beans son "*portables*" a través de contenedores, y pueden ser comprados o construidos.

1.6.2 Tipos de Enterprise Java Beans

Session Bean.- es ejecutado por un solo cliente y es de vida corta, no representa datos en la base de datos de manera directa pero puede acceder y actualizar tales datos. Existen dos clases de session bean:

- Stateless.- una identidad común para los objetos.
- Stateful.- una identidad única para el objeto.

Entity Bean.- compartida entre los clientes. Es de larga-vida, mapea los datos en la base de datos o en la aplicación, mantiene un estado de persistencia que puede ser:

- Container-managed.- acceso definido en el desarrollo. La lógica de la persistencia es delegada al contenedor.
- Bean-managed.- acceso definido como parte del bean. La lógica de la persistencia es implementada dentro del bean.

Aplicaciones EJB usan una mezcla de session bean y entity bean para implementar la lógica del negocio, como lo muestra la siguiente figura:

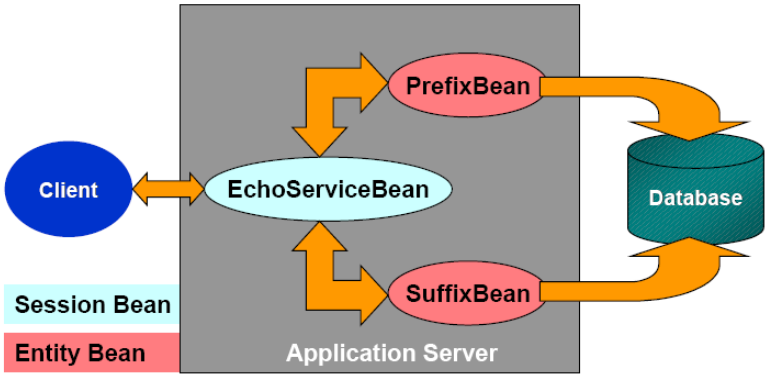


Figura 11. Entity Bean, Session Bean

Uno de los conceptos importantes dentro de EJB es la "*persistencia*", para entender este concepto hacemos uso de dos gráficos importantes:

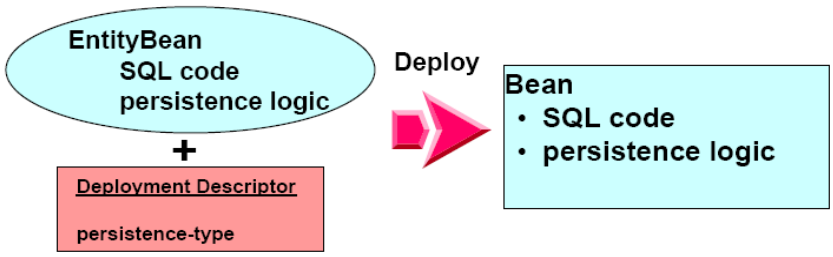


Figura 12. Bean-Managed Persistence

El bean permite actualizar la base de datos haciendo uso del JDBC, SQLJ directamente en el entity bean. El entity bean está atado a la fuente de datos en donde reside la entidad, es más portable a través de plataformas que un container-managed.

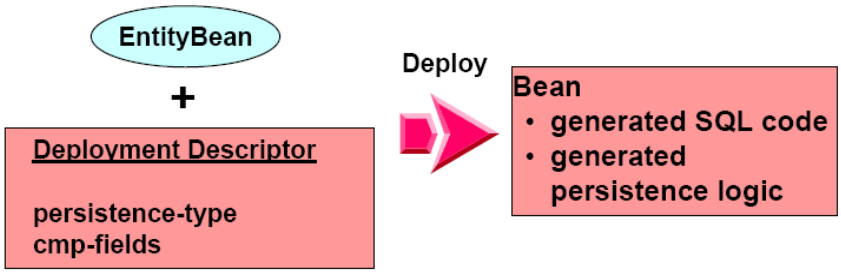


Figura 13. Container-Manged Persistente

Los componentes de acceso a los datos (como JDBC y SQLJ) son desarrollados en la fase de desarrollo por una herramienta contenedora. Entity bean es independiente de la fuente de datos donde reside la entidad.

1.6.3 Pasos para desarrollar una Aplicación EJB

A continuación se presenta una breve descripción del proceso para implementar una aplicación EJB.

1.6.3.1 **Crear las Interfaces.-** Se crea las interfaces para las llamadas locales y remotas de los EJB, para lo cual se definen el ciclo de vida y el proveedor de las interfaces para el cliente que realiza la llamada remota.

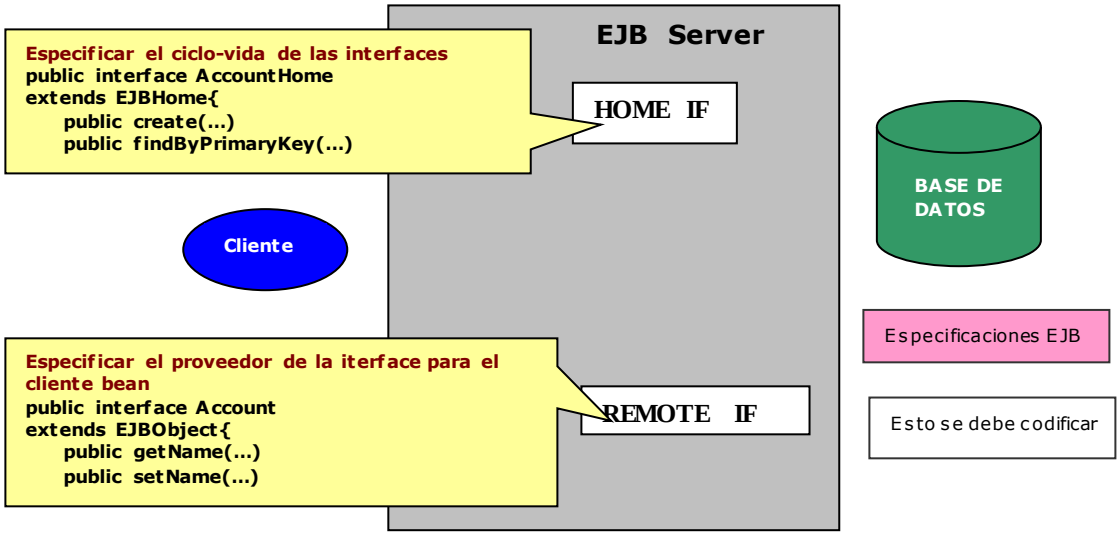


Figura 14. Crear las Interfaces EJB

1.6.3.2 Crear las Implementaciones.- Se crean clases para implementar las interfaces BEAN, en estas clases se tienen que definir las funciones y métodos internos.

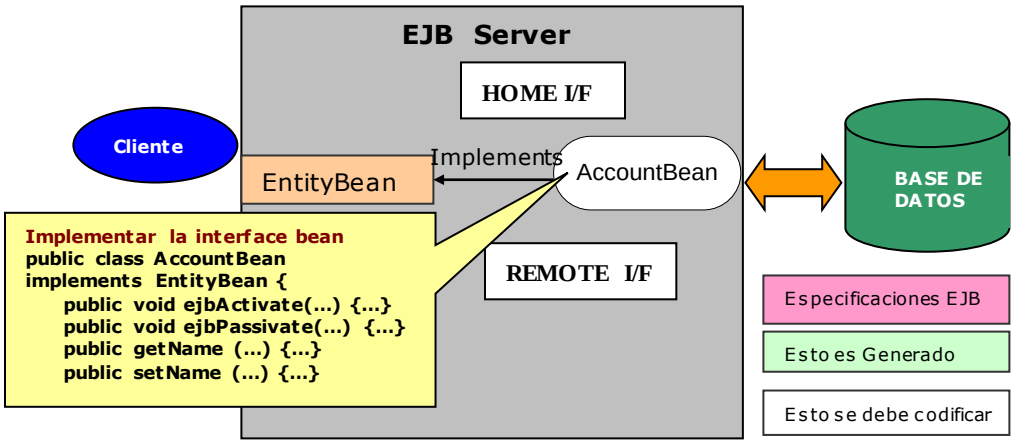


Figura 15. Crear las Implementaciones

1.6.3.3 Deployment Descriptor.- Con la definición de las clases y la plataforma de trabajo lista, se procede a describir los métodos generales tales como Seguridad, Conexión a Base de Datos, etc.

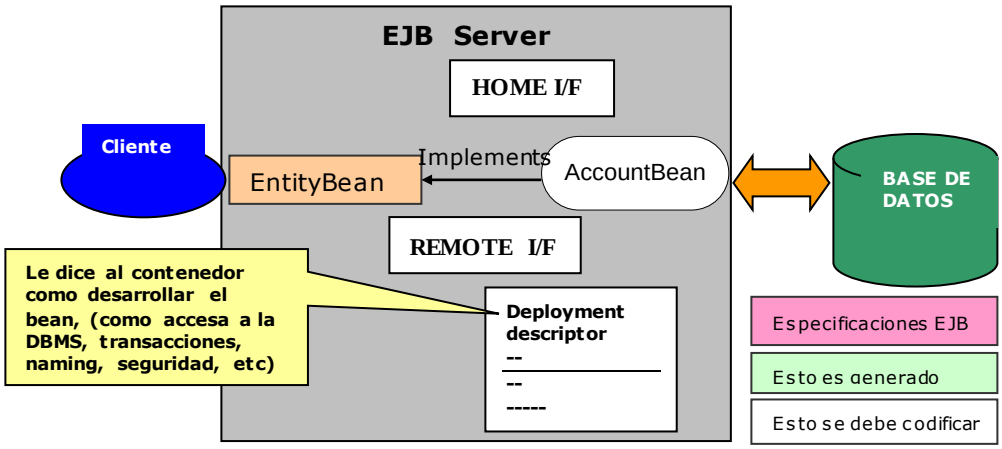


Figura 16. Deployment Descriptor

1.6.3.4 Deploy.- Con todo definido y con las descripciones para desarrollar las clases se las implementa, de tal forma que el cliente puede llamar de forma transparente a un objeto local o a un objeto remoto.

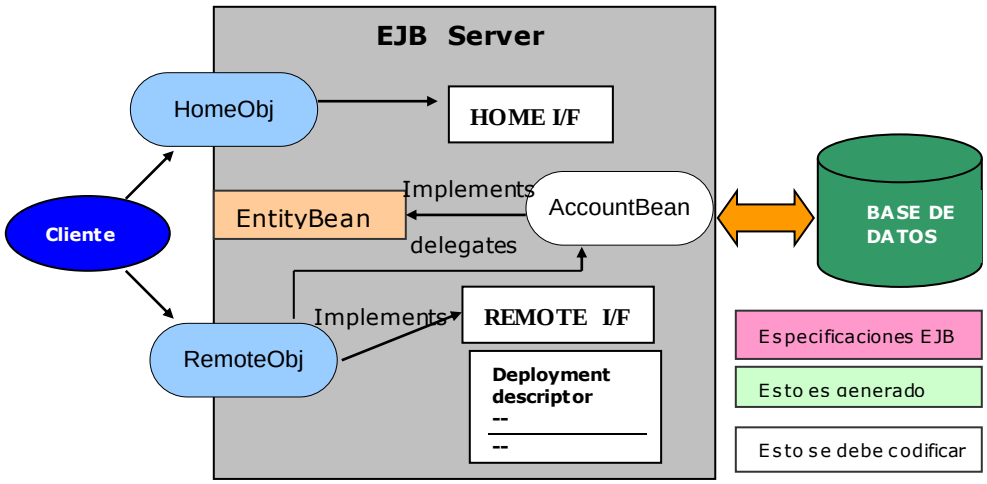


Figura 17. Deploy

1.7 EJB y CORBA

EJB y CORBA son estándares complementarios.

EJB usa CORBA para:

- Habilitar acceso a aplicaciones EJB a clientes no-Java.

- Interoperabilidad para ambientes EJB que incluyen sistemas de múltiples fabricantes.

El mapeo de EJB-CORBA se usa para la separación de especificaciones y son:

- Mapeo de interfaces EJB a RMI-IIOP
- Propagar transaction context. (El objeto del contexto de la transacción permite que los clientes compongan el trabajo de múltiples objetos del servidor de la transacción en una sola transacción, sin tener que desarrollar un nuevo componente específicamente para ese propósito.)
- Preparar security context (Un objeto que encapsula la información compartida del estado con respecto a la seguridad entre dos entidades).
- Interoperabilidad de naming service

Capítulo 2

2 DESCRIPCIÓN DEL PROYECTO

2.1 Objetivos

- Implementar una aplicación usando la arquitectura Cliente/Servidor, que permita realizar las siguientes transacciones: reserva de habitaciones para un hotel, check-In/Out y emisión de la factura a cancelar.
- El sistema debe modelar una aplicación de múltiples capas.

- Demostrar la funcionalidad y la aplicación de CORBA.
- Demostrar la facilidad de obtener los recursos distribuidos a través de Internet.

2.2 Justificación

Se escogió un proyecto con requerimientos transaccionales, que permitan aplicar la nueva tecnología cubriendo todas las necesidades del negocio, mediante el uso de Internet.

2.3 Especificaciones del proyecto

El sistema facilita la reservación de una habitación en un hotel, y provee las siguientes opciones:

- **Reservación de una habitación.-** Se ingresa un nombre, número de cédula/pasaporte, tipo de habitación, fecha de check-in y check-out, y el número de personas. El sistema devuelve el valor que se debe facturar.

- **Cancelación de la reservación.-** Las habitaciones dentro de esta reservación se liberan y quedan disponibles para nuevas reservaciones o check-in.
- **Check-In.-** Se verifica en el sistema las habitaciones disponibles si no hay reservación previa. De existir se asigna la/s habitaciones reservadas.
- **Check-Out.-** El usuario devuelve la habitación. De acuerdo a los días que usó la misma el sistema calcula el costo y emite la respectiva factura. Requiere el id de usuario, nombre, el detalle de las habitaciones, la fecha de check-in y check-out.

Transacciones adicionales:

- **Mantenimiento de Habitaciones.-** Ingresar, modificar, eliminar habitaciones, indicando tipo de habitación, número, estado, capacidad.
- **Mantenimiento de Feriados.-** Ingresar, modificar, eliminar tabla de feriados, con costos y periodos.
- **Reportes.-** Listado de habitaciones indicando el estado, listado de empleados, listado de clientes, entre otros.

El sistema cuenta con una base de datos inicial, para las respectivas pruebas y operación.

2.4 Descripción general del proyecto

El aplicativo usa una arquitectura Cliente/Servidor, haciendo uso de CORBA, JDBC, y Applets. Se creó una aplicación de tres capas, accesible a través del Web. En definitiva el sistema presenta los Applets que serán invocados desde una página HTML que es cargada desde el Servidor Web.

Los clientes Applets invocan a objetos en el servidor (objetos CORBA), a través del protocolo IIOP. Toda la lógica del negocio se encuentra en los objetos servidores y estos almacenan los datos en la base de datos SQL con soporte JDBC.

Las páginas HTML y los Applets deben ubicarse en la maquina donde se ejecuta el servidor web. Los objetos servidores CORBA pueden encontrarse en el mismo servidor o en otro en la LAN. La base de datos puede residir de igual manera en el servidor web o en otro equipo en la LAN.

2.4.1 Detalles del Cliente

Parte de la lógica de la aplicación reside en el cliente. En nuestro caso el Applet realiza invocación de métodos CORBA, por lo cual el objeto servidor espera de manera pasiva los requerimientos de los clientes para comenzar la ejecución de sus métodos.

El GUI de los clientes se implementa totalmente usando Applets. Dentro de la lógica del Applet se encuentran las siguientes funciones:

- Validar los datos que se ingresen.
- Invocar a funciones de cálculos.
- Invocar a métodos en el lado del servidor, para que ejecuten procedimientos o transacciones a la base de tatos.
- Presentar los resultados en los mismos GUIS que hicieron la invocación.

2.4.2 Detalles del Servidor

El programa servidor de la aplicación fue desarrollado totalmente en lenguaje Java haciendo uso de las ventajas que ofrece la arquitectura CORBA. El principio es básicamente el uso de métodos o funciones de "Objetos" CORBA que van ser invocados desde un Applet, el cual puede ejecutarse desde cualquier Navegador. Esta invocación se encuentra embebida en el Applet e inicia acciones en el lado del ambiente servidor como.

- Levantar conexiones a la base de datos para cada cliente.
- Realizar operaciones y cálculos específicos sobre los datos de la base.

2.4.3 Procesos involucrados en el Sistema

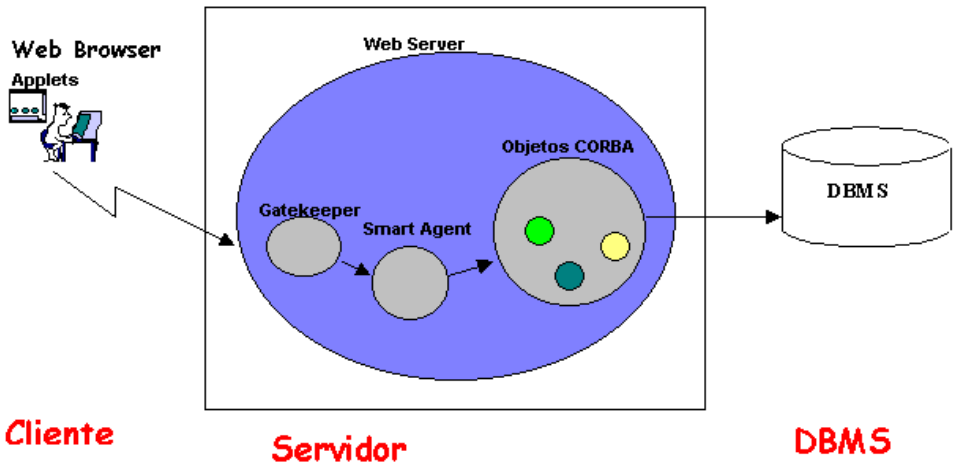


Figura 18. Procesos involucrados en el Sistema

Ambiente Cliente, como podemos apreciar en la figura anterior, en este ambiente se tienen a los Applets de Java (procesos clientes) y al web browser donde se ejecutan estos Applets.

Ambiente Servidor, Web Server, el Smart Agent, el Gatekeeper, y el proceso servidor desarrollado en Java (alberga los objetos CORBA). Son los procesos que atienden los requerimientos de los clientes.

El servidor WEB atiende los requerimientos hechos usando el protocolo HTTP y permite la descarga de las páginas HTML en los browsers de los clientes.

VisiBroker's osagent (**Smart Agent**) es un proveedor de servicios de directorios dinámico. Este provee la facilidad tanto a las aplicaciones clientes y a los objetos implementados. Cuando una aplicación cliente invoca al método bind sobre un objeto, el osagent localiza la implementación específica y al objeto para poder establecer una conexión entre el cliente y la implementación. Objetos de la implementación registran sus objetos con el osagent para que las aplicaciones clientes puedan localizar y usar estos objetos. Cuando un objeto o implementación es destruido, el osagent lo remueve de la lista de objetos disponibles.

VisiBroker **Gatekeeper** proporciona estrategias para el uso de mensajes UDP de broadcast para localizar procesos osagent. Cuando un osagent es encontrado se usa para todas las subsiguientes interacciones. Si un osagent no puede ser encontrado por cualquiera de estas alternativas, el ORB volverá a utilizar el esquema emisor del mensaje para localizar un osagent.

EL VisiBroker **Gatekeeper** permite a los programas clientes publicar la operación de los objetos que residen en el Web Server y siempre recibir requerimientos provenientes de estos objetos, conformándose a las restricciones de seguridad impuestas por los web

browsers. El Gatekeeper también maneja la comunicación en un firewall y se puede usar como servicio HTTP.

2.4.4 Detalles de la Base de Datos

La persistencia de la clase consiste en que sus datos puedan perdurar en tiempo, esta persistencia puede ser en memoria, archivos textos o una base de datos.
En este proyecto se utiliza una base de datos relacional, este modelo permite que cada objeto guarde su información en la base de datos.

Generalmente se mapea una clase a una tabla, pero esto depende del desempeño que se quiera dar a la solución, se debe tomar en cuenta que a medida que crece el número de clases, crece el número de tablas y relaciones. El sistema aumenta su complejidad para la recuperación y actualización de datos.

Existen dos formas de implementar la persistencia.

- Que cada clase tenga incorporados metidos para acceder a la base de datos y manipular su información. Este método es fácil de implementar pero difícil para darle mantenimiento.

- Que exista un conjunto de clases que controlen la persistencia de los objetos, son las clases que hacen de interface entre los objetos y la base de datos. Este método es difícil de implementar pero muy practico para su mantenimiento.

Para este proyecto se utilizo el segundo método, una sola clase que controle el acceso a la base de datos y se encargue de la persistencia de los objetos que lo requieran.

Para que un objeto sea persistente, el mismo debe heredar de la clase Persistencia. Es decir, este objeto adquiere el comportamiento de la clase, con lo cual aprende a instanciarce desde la base de datos, guardarse, actualizarse, mantener estados, etc.

Este modelo inicialmente se conecta a una Base de Datos, luego traduce el comportamiento de un objeto y lo lleva a sentencias SQL por medio de un conjunto de métodos que mapean uno a uno los atributos de la clase a columnas de una tabla.

CAPÍTULO 3

3 ANÁLISIS Y DISEÑO DEL SISTEMA

3.1 Introducción

El análisis y diseño del sistema fue realizado con la metodología de Análisis y Diseño Orientación a Objetos, este análisis se inicio con la metodología O.M.T. de Rumbaugh, pero al final se utilizó UML que es el Lenguaje de Modelamiento Unificado que fue desarrollado por la empresa RATIONAL. Esta empresa fue fundada por Rumbaugh, Booch y Jacobson. El UML fue aceptado como estándar por la OMG en

1997. Desde entonces hasta la actualidad ha sufrido varias actualizaciones como estándar. La empresa RATIONAL fue adquirida por I.B.M.

Se comenzó levantando los requerimientos del Sistema a través del modelamiento dinámico. Es decir llevar los requerimientos externos descritos en los casos de uso a requerimientos internos que es el modelamiento de objetos en función de los cambios en el tiempo.

Describimos como se comportan los objetos. Tomamos los casos de uso que describen el comportamiento del sistema a alto nivel para definir los escenarios que describen la funcionalidad de cada uno de los casos de uso, y por medio de estos encontramos los objetos que actúan en estos escenarios para luego hacer el análisis de diseño de interacción de objetos y el diagrama de estados.

3.2 Análisis de requerimientos

En el Análisis y Diseño de nuestro proyecto se ha definido el siguiente alcance:

Límites del Sistema

Reservación

Descripción: En esta parte del proceso se involucra todo lo referente a: Reservación de una habitación en línea, dentro de la disponibilidad de habitaciones del hotel y que cumpla con las necesidades del cliente.

Actores: Cliente

Administración

Descripción: En esta parte del proceso, se realiza parte operativa que involucra directamente al HOTEL, como son: Check-In, Check-Out, Ingreso y modificaciones de empleados, Ingreso y modificación de feriados, Ingreso y modificación de habitaciones, tipos de habitaciones, además de consultas y reportes.

Actores: Administrador

Recepcionista

3.3 Casos de Uso

3.3.1 Lista de casos de Uso

1) Reservar Habitación.
2) Cancelar Reservación.
3) Check-In con Reserva.
4) Check-In sin Reserva.
5) Check-Out.
6) Ingresa Habitación.
7) Ingreso de Tipo de Habitación.
8) Ingreso de Feriado.
9) Consulta Habitación.

3.3.2 Diagrama de Casos de USO

Aquí se definen las principales características del sistema "Reservación de habitaciones de un hotel", además se definen los actores que interactúan con el sistema. Toda esta relación separada por los límites del sistema.

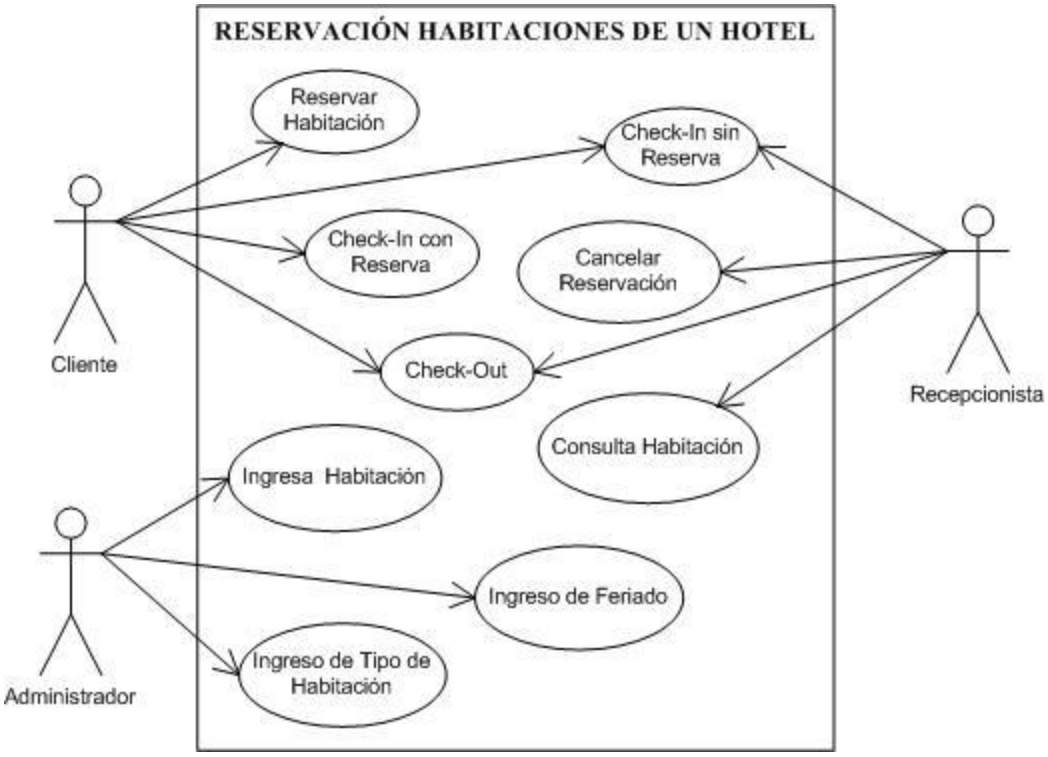


Figura 19. Casos de Uso

3.3.3 Descripción de Casos de Uso

1- Reservar Habitación

Definición	Un cliente desea reservar una habitación. Si se ingresa correctamente la información, una reservación es registrada a nombre del cliente, y se le devuelve un número de reservación. La reservación incluye la validación de la disponibilidad de habitaciones de acuerdo a la exigencia del cliente.
Notas:	• Cada reservación es por habitación. • Requiere información del cliente: número de cédula o de pasaporte, nombre y apellido, dirección de domicilio, teléfono, país de residencia, correo electrónico y empresa a la que pertenece si se encuentra en una empresa de trabajo. • Requiere información de las características de la habitación, entre otras: fecha de ingreso al hotel, numero de días para reservar, numero de personas que se hospedarán y el tipo de habitación. • Si no existen habitaciones que cumplan con las exigencias, no se realiza la reservación. • Se le notifica al cliente él número de reservación obtenida.

Actores:	• Cliente
Objetos:	• Habitación • Reservación • Tipo de Habitación • Feriado • Cliente

2- Cancelar Reservación

Definición:	El recepcionista consulta una determinada habitación, si la habitación está reservada y la fecha ya pasó el recepcionista tiene que eliminar la reservación.
Notas:	• Sólo se puede eliminar la reservación por dos motivos: en caso de llegada la fecha de ingreso el cliente no se ha acercado a realizar un Check-in, o si el cliente solicita la cancelación de su reservación. • Se pude cancelar una reservación de una habitación que se encuentre disponible.
Actores:	• Cliente • Recepcionista

3- Check-In con Reserva

Definición:	Un cliente solicita una habitación que ha reservado a su nombre previamente. Debe existir una reservación a nombre del cliente, debe indicar su numero de reservación, si se tiene éxito una habitación queda registrada a nombre del cliente y se le entrega la llave de la habitación y el número de la misma.
Notas:	• El cliente debe presentarse a solicitar el Check-In, en la fecha que ingresó en la reservación previa. • Cada Check-In es por habitación. • El Check-In se hace a nombre del solicitante. • EL cliente solicita el Check-In, con su nombre o número de reservación. • Solo en el Check-In se puede modificar el número de días de hospedaje. • Se escoge la habitación de las habitaciones disponibles. • Se le notifica al cliente el número de habitación, y el costo tentativo, de acuerdo al número de días de hospedaje. • El valor a pagar se calcula de acuerdo al precio de ese tipo de habitación y al número de personas que se hospedan en ella.

	• La habitación que es entregada al cliente, quedará como Ocupada.
Actores:	• Cliente • Recepcionista
Objetos:	• Check-In • Reservación • Habitación • Cliente • Feriados • Tipo de Habitación

4- Check-In sin Reserva

Definición:	El cliente solicita una habitación, sin existir una reservación previa, se verifica si existen habitaciones disponibles. Si se tiene éxito la habitación queda registrada a nombre del cliente y se le entrega la llave de la habitación.

Notas:	• Cada Check-In es por habitación. • Requiere información del cliente: número de cédula o de pasaporte, nombre y apellido, dirección de domicilio, teléfono, país de residencia, correo electrónico y empresa a la que pertenece si la habitación será cancelada por la empresa. • Requiere información de las características de la habitación, entre otras: fecha de ingreso al hotel, número de días para reservar, número de personas que se hospedarán y el tipo de habitación. • Se escoge la habitación entre las habitaciones disponibles. • Se le notifica al cliente el número de habitación, y el costo tentativo, de acuerdo al número de días de hospedaje. • El valor a pagar se calcula de acuerdo al precio, de ese tipo de habitación y al número de personas que se hospedan en ella. • La habitación que es entregada al cliente, quedará como Ocupada.
Actores:	• Cliente • Recepcionista
Objetos:	• Check-In • Reservación

	• Habitación • Cliente • Feriado • Tipo de Habitación

5- Check-Out

Definición:	El cliente entrega la llave de la habitación, y solicita la cuenta. Se verifica los datos de Check-In y se registra el Check-Out, se recepta la llave y se le notifica al cliente el monto a facturar.
Notas:	• En el momento del Check-Out se registra el número de días de hospedaje, y el número de personas. • Cada Check-Out es por habitación. • La habitación entregada por el cliente, queda como disponible.
Actores:	• Cliente • Recepcionista
Objetos:	• Check-In • Reservación • Habitación • Cliente

	• Feriado • Tipo de Habitación

6- Ingresa Habitación

Definición:	Si físicamente una habitación se ha habilitado para su uso, se debe registrar como disponible.
Notas:	• Se debe especificar a que categoría pertenece la habitación. • Se debe ingresar la capacidad en número de personas • Se registra como disponible. • Se debe indicar el número de habitación que la identificara.
Actores:	• Administrador
Objetos:	• Habitación. • Tipo de Habitación.

7- Ingresa Tipo de Habitación

Definición:	El administrador ingresa un nuevo tipo de habitación, que identifique un grupo de habitaciones con características similares.

Notas:	• Se debe definir por cada tipo de habitación un nombre que las identifique. • Se debe definir por cada tipo, el costo diario. • Se debe definir por cada tipo, las características comunes de este tipo de habitación.
Actores:	• Administrador
Objetos:	• Tipo de Habitación

8- Ingreso de Feriado

Definición:	El administrador registra un feriado o temporada, que se define como un periodo de tiempo, durante el cual se puede tener descuentos o recargos, sobre el costo de las habitaciones.
Notas:	• Se debe definir por cada feriado, una fecha inicial y final • Un porcentaje de descuento o recargo debe definirse por cada feriado. • EL nombre que identifique el feriado, ejemplo Carnaval.
Actores:	• Administrador
Objetos:	• Feriados • Empleado

9- Consulta de Habitación

Definición:	El recepcionista o administrador consultan las habitaciones que se encuentran registradas. Se muestra información general.
Notas:	• Muestra información de las habitaciones como: número de habitación, categoría a la que pertenece, y el estado actual en la que se encuentran
Actores:	• Administrador • Recepcionista
Objetos:	• Tipo de Habitación • Habitación • Empleado

3.3.3 Lista de Actores

Cliente.- Una persona que desea contratar el servicio ofrecido, o pagar por el servicio prestado. Se convierte en huésped una vez hospedado. Solicita reservaciones, Check-In, y Check-Out de habitaciones.

Recepcionista.- La persona que trata directamente con el cliente y se encarga de satisfacer las dudas y necesidades a un cliente. Se encargara de realizar reservaciones,

check-in, check-out solicitados por el cliente. También se encargará de registrar los datos necesarios del cliente.

Administrador.- Se encarga de supervisar la parte administrativa del hotel. Decide sobre precios, porcentajes, feriados, nuevas habitaciones y tipos de habitaciones.

3.4 Definición de Escenarios

La definición de los escenarios son los diferentes flujos que se controlan para cada caso de uso, todo escenario tiene un inicio y un fin para lo cual se requieren asunciones (Pre-Condiciones) y resultados (Post-Condiciones).

En este documento se definen dos escenarios por cada caso de uso, el escenario exitoso (Flujo Principal) y el escenario no exitoso más común (Flujo Alternativo).

Escenarios del caso de Uso 1: Reservar de Habitación

Procedimiento: El cliente registra una reservación, entregando sus datos personales y selecciona las características deseadas de la habitación que se reservara.

3.4.1.1 El Cliente reserva una habitación exitosamente.

3.4.1.2 No existen habitaciones disponibles. No se reserva la habitación.

3.4.1.3 No existen habitaciones con la capacidad solicitada. No se reserva la habitación.

3.4.1.4 No existen habitaciones disponibles con el tipo solicitado. No se reserva la habitación.

Escenario 3.4.1.1: El Cliente reserva una habitación exitosamente..

Asunciones:

1. Debe existir por lo menos una habitación disponible, con las características solicitadas.
2. El Cliente ingresa sus datos de manera correcta

Resultados:

1. Cliente reserva habitación exitosamente.
2. Se registra en el sistema una reservación.
3. Se genera un número de reservación.

Escenario 3.4.1.2: El Cliente no reserva la habitación, porque no existen habitaciones disponibles en todo el hotel.

Asunciones:

1. El cliente solicita un tipo de habitación específico (Presidencial).
2. El cliente solicita una capacidad determinada (5).
3. No existen habitaciones disponibles que satisfagan las exigencias de la reservación que el cliente solicita. El hotel esta lleno.

Resultados:

1. No se registra una reservación.

Escenario 3.4.1.3: El Cliente no reserva la habitación, porque no existen habitaciones con la capacidad solicitada.

Asunciones:

1. El cliente solicita un tipo de habitación específico (Presidencial).
2. El cliente solicita una capacidad determinada (5).
3. No existen habitaciones con la capacidad solicitada.

Resultados:

1. No se registra una reservación.
2. El cliente puede realizar una nueva reservación cambiando la capacidad de personas.

Escenario 3.4.1.4: El Cliente no reserva la habitación, porque no existen habitaciones disponibles con el tipo solicitado.

Asunciones:

1. El cliente solicita un tipo de habitación específico (Presidencial).
2. El cliente solicita una capacidad determinada (5).
3. No existen habitaciones el tipo solicitado.

Resultados:

1. No se registra una reservación.
2. El cliente puede realizar una nueva reservación cambiando el tipo de habitación.

Escenarios del caso de Uso 2: Cancelar Reservación

Procedimiento: Por medio del número de reservación, se podrá consultar y eliminar dicha reservación. Se puede consultar las reservaciones cuya fecha inicial sea menor a la fecha de consulta.

3.4.1.5 Cancelación de reservación exitosa

3.4.1.6 Cancelación de reservación sin éxito. Porque no Existe.

Escenario 3.4.1.5: Cancelación de reservación exitosa

Asunciones:

1. Existe una reservación con él número que se desea eliminar.
2. Existe la solicitud para eliminar una reservación.

Resultados:

1. EL recepcionista elimina la reservación de manera exitosa.
2. Se elimina el registro de reservación.

Escenario 3.4.1.6: Cancelación de reservación sin éxito. Porque no existe número de reservación

Asunciones:

1. Existe una solicitud para eliminar una reservación.
2. Número de reservación no existe en el sistema.

Resultados:

1. No se elimina ningún registro.
2. Se muestra un mensaje del sistema

Escenarios del caso de Uso 3: Check-In con Reserva

Procedimiento: Consiste en registrar el check-in de una habitación a nombre de una persona que previamente ha realizado una reservación.

3.4.1.7 Check-In de una habitación con reservación exitosa.

3.4.1.8 Check-In de una habitación con reservación sin éxito. No existe reservación previa

Escenario 3.4.1.7: Check-In de una habitación con reservación exitosa.

Asunciones:

1. Existe una reservación a nombre del cliente, que solicita el check-in.
2. El cliente solicita hacer un Check In.

Resultados:

1. El cliente realiza el check-in de la habitación reservada, exitosamente.
2. Se registra un check-in a nombre del cliente, de la habitación reservada.
3. Se actualiza el estado de la habitación, la habitación pasa al estado ocupado.
4. Se obtiene el número de habitación que se entregara al cliente.

Escenario 3.4.1.8: Check-In de una habitación sin éxito por no contar con una reservación previa.

Asunciones:

1. El cliente solicita hacer un Check In.
2. No existe reservación registrada.

Resultados:

1. No se registra el check-in de la habitación
2. El cliente puede realizar un Check In sin reserva.
3. El cliente puede hacer una Reserva.

Escenarios del caso de Uso 4: Check-In sin Reserva

Procedimiento: Consiste en registrar el check-in de una habitación a nombre de una persona sin existir la reservación por parte del cliente.

3.4.1.9 El cliente solicita el check-in de manera exitosa.

3.4.2.0 El cliente solicita el check-in sin éxito, porque no existe habitación disponible.

Escenario 3.4.1.9: El cliente solicita e check-in de manera exitosa.

Asunciones:

1. El cliente solicita una habitación, y no tiene una reservación.
2. Existe por lo menos una habitación disponible, para registrar el check-in a nombre del cliente.
3. Se ingresa correctamente la información del cliente.

Resultados:

1. El cliente realizó el check-in, de la habitación, exitosamente.
2. Un check-in a nombre del cliente es registrado.
3. Una habitación es ocupada.
4. Número de habitación que se le realizó un check-in.

Escenario 3.4.2.0: El cliente solicita el check-in sin éxito, porque no existe habitación disponible

Asunciones:

1. El cliente solicita una habitación, pero no tiene una reservación.
2. No existen habitaciones disponibles para realizar el check-in.
3. Se ingresa correctamente la información del cliente.

Resultados:

1. No se ingresa el check-in
2. El cliente no puede volver a solicitar el check-in.

Escenarios del caso de Uso 5: Check-Out

Procedimiento: Consiste en registrar la salida del cliente y facturar el uso de la habitación por el periodo de presencia.

3.4.2.1 El cliente realiza el check-out de la habitación de manera exitosa

3.4.2.2 El cliente solicita el check-out de la habitación sin éxito, porque no tiene un Check In registrado a su nombre

Escenario 3.4.2.1: El cliente realiza el check-out de la habitación de manera exitosa.

Asunciones:

1. Existe registrado un check-in, a nombre del cliente que solicita el check-out.
2. El cliente acepta pagar el monto adeudado.

Resultados:

1. El cliente realiza un check-out exitoso de la habitación.
2. Se registra un check-out a nombre del cliente que lo solicita.
3. Habitación ocupada pasa a estar disponible.
4. Se calcula el monto que debe pagar el cliente.

Escenario 3.4.2.2: El cliente realiza el check-out de la habitación sin éxito. Porque no tiene un Check In registrado a su nombre

Asunciones:

1. No existe un check-in a nombre del cliente que solicita el check-out.
2. El cliente solicita el Check Out.

Resultados:

1. No se registra el check-out a nombre del cliente.
2. No se puede hacer el cobro del cliente.

Escenarios del caso de Uso 6: Ingresa Habitación

Procedimiento: Consiste en ingresar una nueva habitación al sistema.

3.4.2.3 El administrador ingresa la nueva habitación exitosamente.

3.4.2.4 No se ingresa la habitación porque ya existe el número asignado.

Escenario 3.4.2.3: El administrador ingresa una nueva habitación exitosamente con su respectivo número, tipo, capacidad y características.

Asunciones:

1. El número de habitación que se ingresa no existe en el sistema.
2. Los tipos de habitación están definidos
3. El administrador conoce las características de la habitación.

Resultados:

1. Se incrementa una habitación en el sistema.
2. EL estado de la habitación es Disponible (“D”).

Escenario 3.4.2.4: El administrador ingresa una nueva habitación sin éxito. Porque ya existe el número de habitación en el sistema.

Asunciones:

1. El numero de habitación que se ingresa ya existe en el sistema.
2. Los tipos de habitación están definidos
3. El administrador conoce las características de la habitación.

Resultados:

1. No se ingresa la habitación.

Escenarios del caso de Uso 7: Ingreso de Tipo de Habitación

Procedimiento: Consiste en ingresar un nuevo tipo de habitación.

3.4.2.5 El administrador ingresa un nuevo tipo de habitación exitosamente.

3.4.2.6 No se ingresa el tipo de habitación porque el precio no es un valor numérico.

Escenario 3.4.2.5: El administrador ingresa un nuevo tipo de habitación con éxito.

Asunciones:

1. El administrador tiene los datos del tipo de habitación: nombre, descripción, costo por persona y las observaciones.

Resultados:

1. Se ingresa un nuevo tipo de habitación en el sistema.

Escenario 3.4.2.6: No se ingresa el nuevo tipo de habitación, porque el costo por persona no es un valor numérico

Asunciones:

1. El administrador tiene los datos del tipo de habitación: nombre, descripción, costo por persona y las observaciones.
2. Se ingresa el costo por persona con el signo de dólares.

Resultados:

1. No se ingresa el tipo de habitación.

Escenarios del caso de Uso 8: Ingreso de Feriado

Procedimiento: Consiste en definir un nuevo feriado para el sistema.

3.4.2.7 El administrador ingresa un nuevo feriado exitosamente.

3.4.2.8 No se ingresa el feriado porque el rango de fechas ya existe como feriado.

Escenario 3.4.2.7: El administrador ingresa el nuevo feriado de forma exitosa.

Asunciones:

1. El administrador tiene definido: el nombre del feriado, porcentaje y si es un incremento o decremento.
2. Se tiene definido la fecha de inicio y fin del feriado, estas fechas no existen en el sistema.

Resultados:

1. Se ingresa un nuevo feriado.

Escenario 3.4.2.8: No se ingresa el feriado, porque el rango de fechas ingresado ya existe en el sistema.

Asunciones:

1. El administrador tiene definido: el nombre del feriado, porcentaje y si es un incremento o decremento.
2. Se tiene definido la fecha de inicio y fin del feriado, estas fechas existen en el sistema.

Resultados:

1. No se ingresa el feriado.

Escenarios del caso de Uso 9: Consulta Habitación

Procedimiento: Consiste en consultar las habitaciones del hotel en base a su estado.

3.4.2.9 Se muestra el listado de las habitaciones que pertenecen al estado seleccionado exitosamente.

3.4.3.0 No existen habitaciones que estén es el estado seleccionado.

Escenario 3.4.2.9: EL recepcionista realiza la consulta de habitaciones para un estado determinado, se muestra el listado exitosamente.

Asunciones:

1. Los estados de las habitaciones están definidos en el sistema.
2. Existen habitaciones para el estado seleccionado.

Resultados:

2. Se muestra el listado de las habitaciones que pertenecen al estado seleccionado.

Escenario 3.4.3.0: No se realiza la consulta exitosamente, porque no existen habitaciones para el estado seleccionado.

Asunciones:

1. Los estados de las habitaciones están definidos en el sistema.
2. No existen habitaciones para el estado seleccionado.

Resultados:

1. Se muestra el listado en blanco, sin registros.

3.5 Diagrama de Interacción de Objetos (DIOs)

Los diagramas de interacción de objetos representan el escenario exitoso de los caso de uso, además se presentan los diagramas de los escenarios no exitosos mas relevantes.

Reserva Habitación Exitosamente

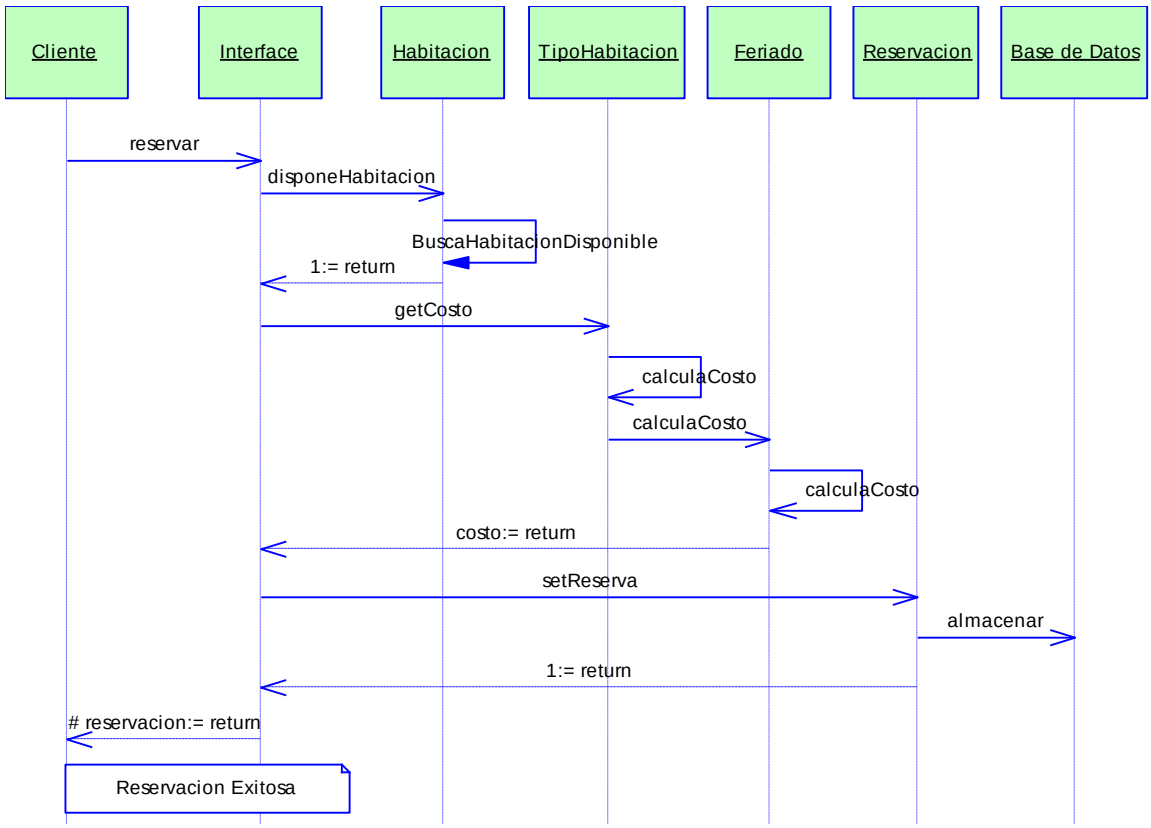


Figura 20. Reserva habitación exitosamente.

Reserva Habitación Fallida

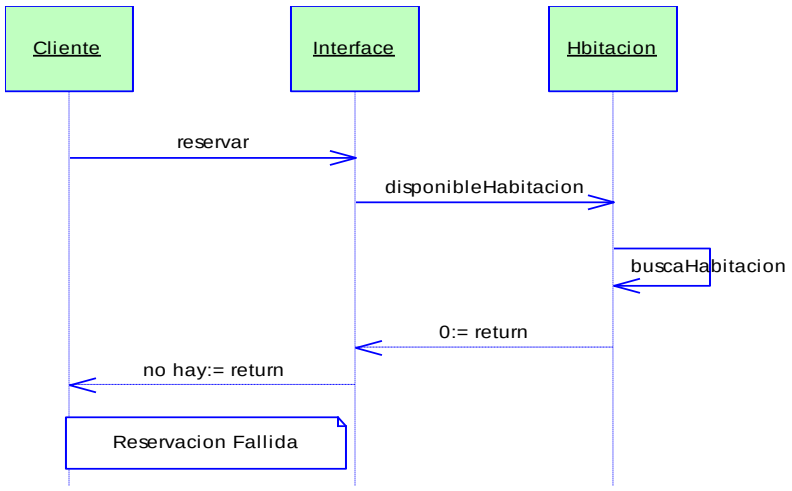


Figura 21. Reservación sin éxito.

Cancelar Reservación

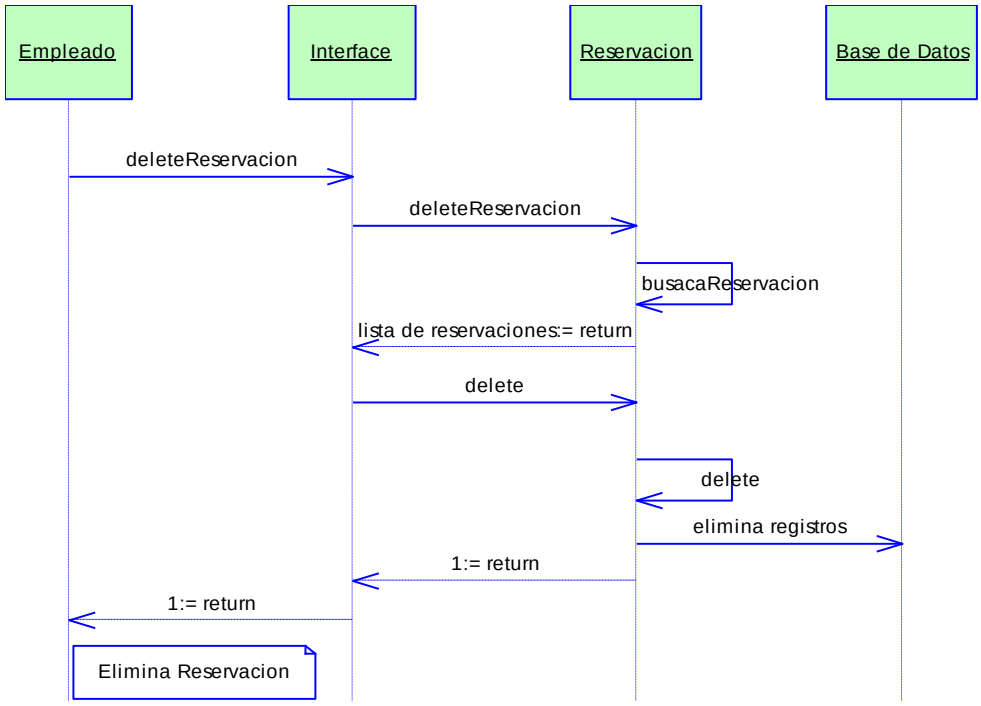


Figura 22. Cancelación de Reservación

Check-In con Reserva.

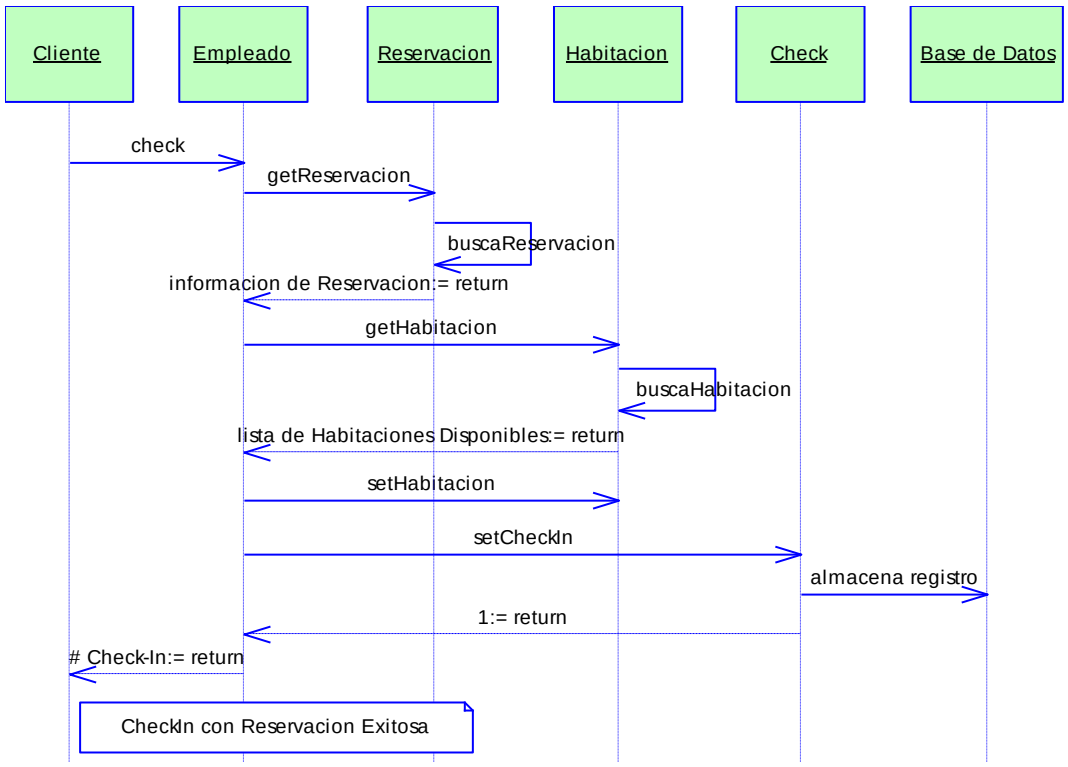


Figura 23. Check-In con reserva exitosa

Check-In sin Reserva.

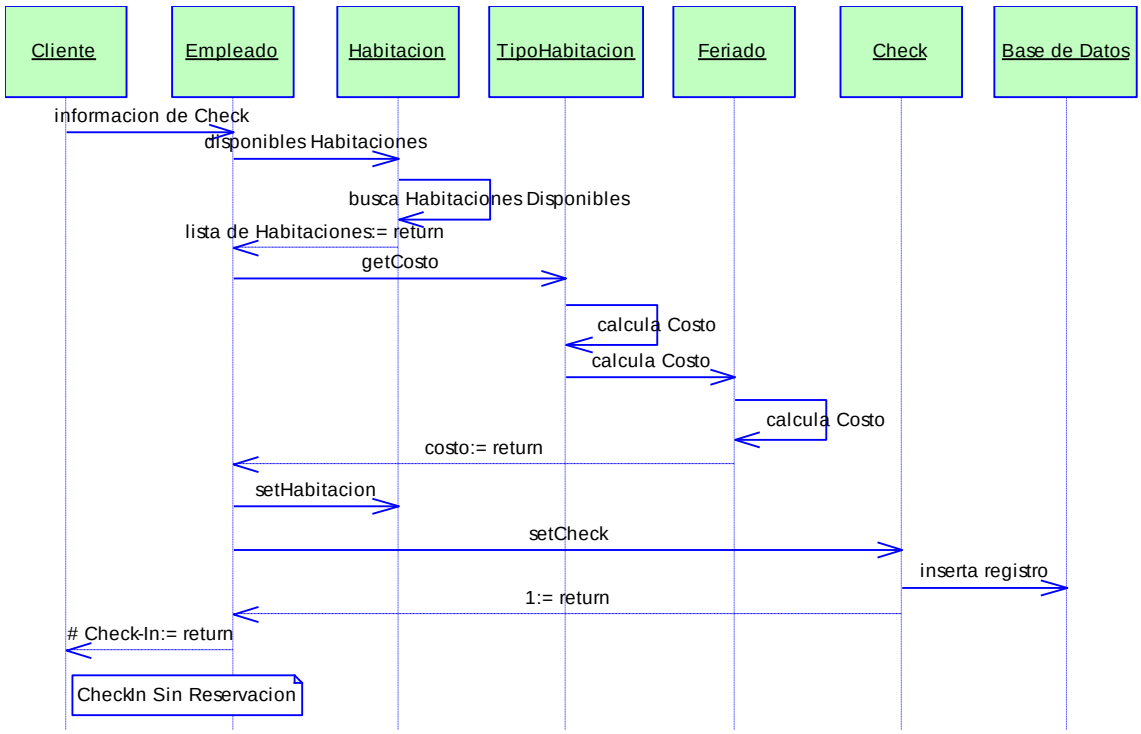


Figura 24. Check-In sin reservación.

Check-Out.

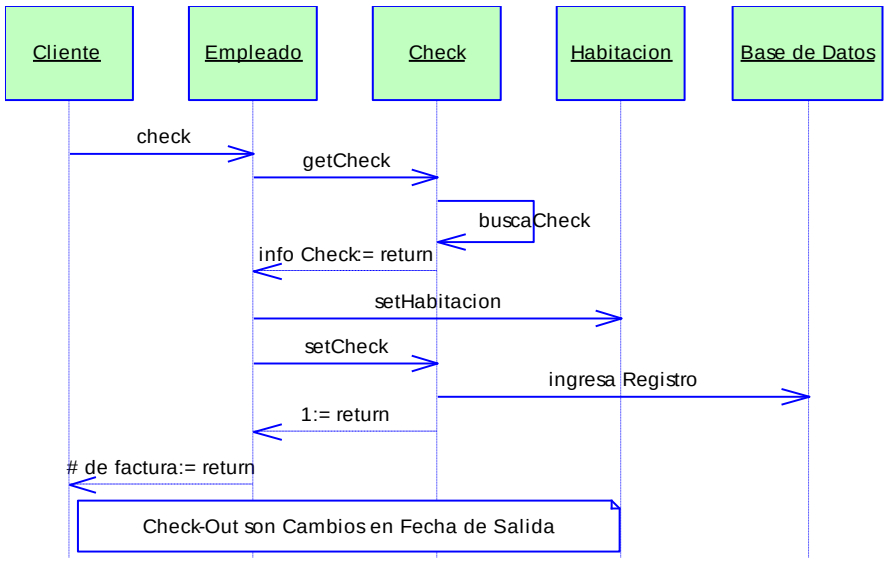


Figura 25. Check-Out sin cambios en la fecha de salida.

Check-Out (Con cambios en la fecha de salida).

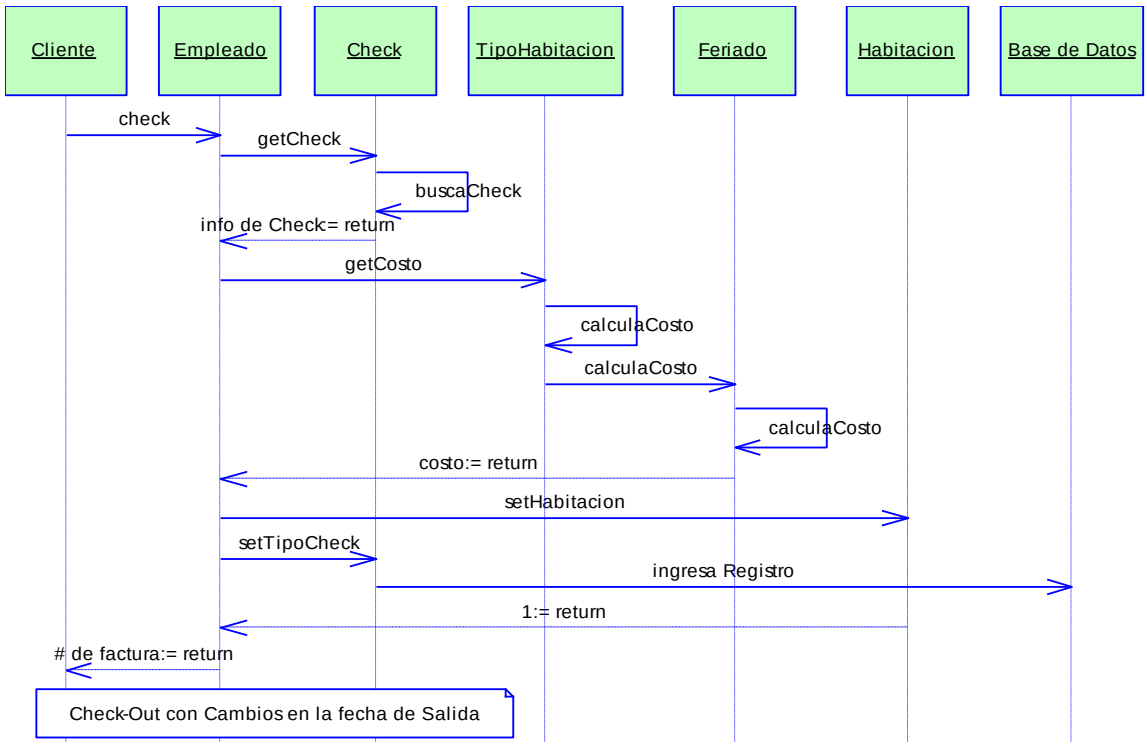


Figura 26. Check-Out con cambios en la fecha de salida.

Ingresa Habitación (exitosamente).

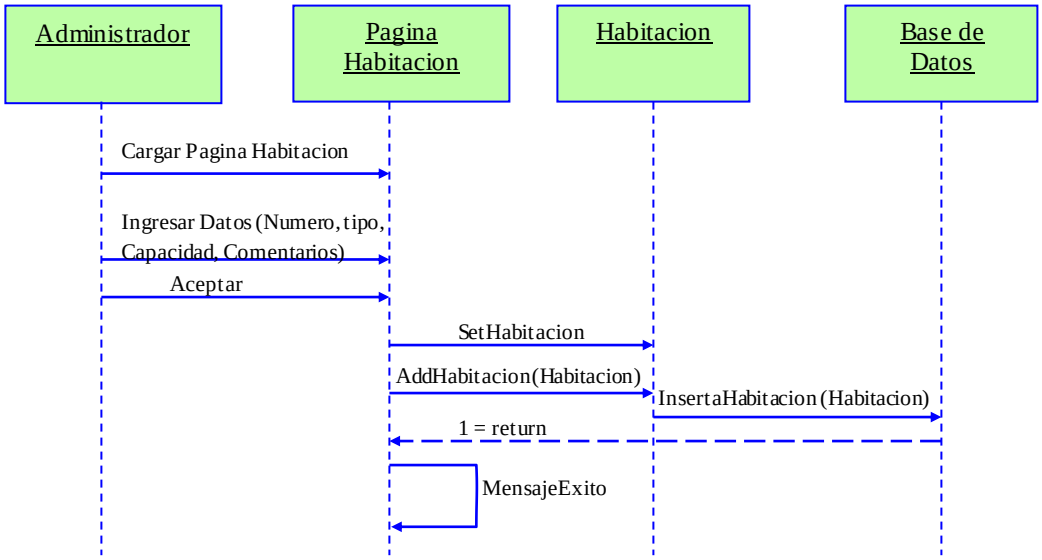


Figura 27. Ingreso de habitación exitosamente.

Ingresa Habitación (sin éxito, porque el numero de habitación ya existe).

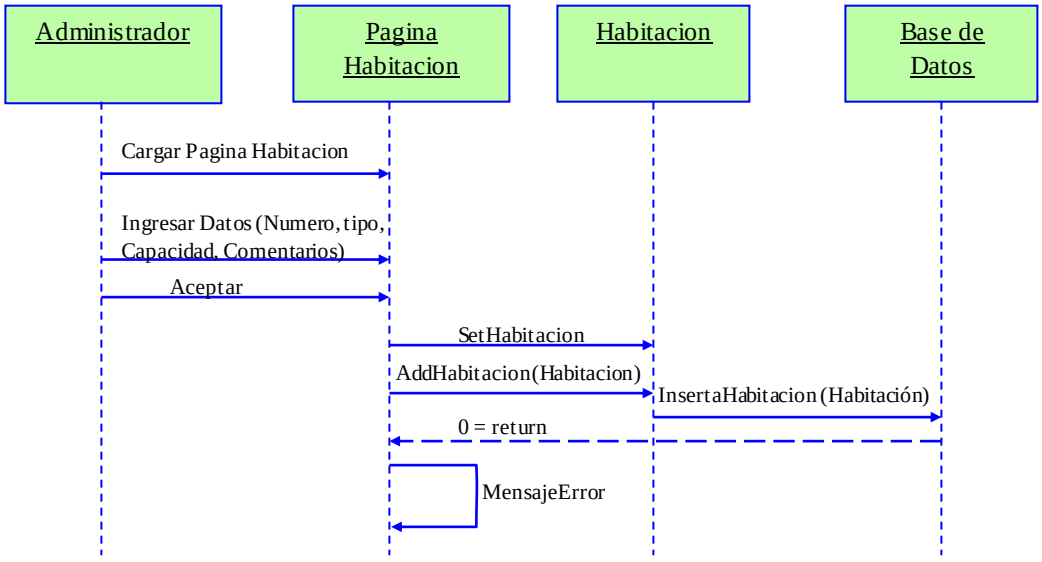


Figura 28. Ingreso de habitación sin éxito.

Ingresa de Tipo de Habitación (exitosamente).

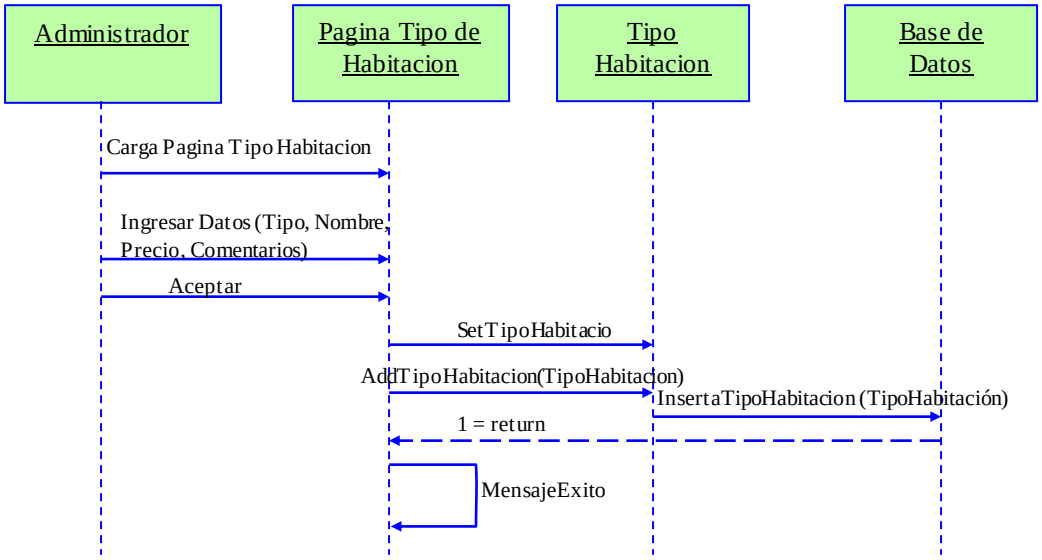


Figura 29. Ingreso de tipo de habitación Exitosamente.

Ingresa Tipo de Habitación (sin éxito, porque el precio no es numérico).

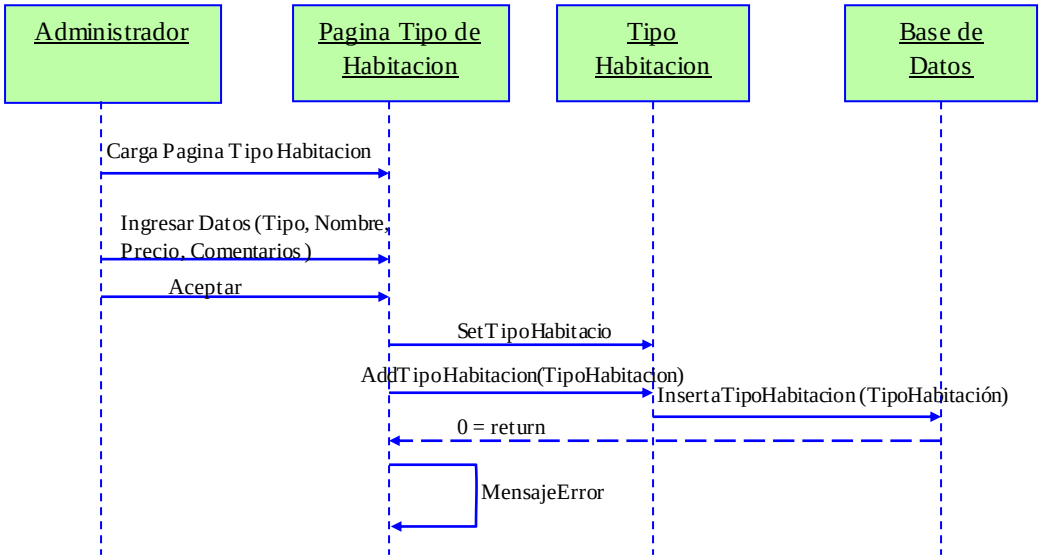


Figura 30. Ingreso de tipo de habitación sin éxito.

Ingresa de Feriado.

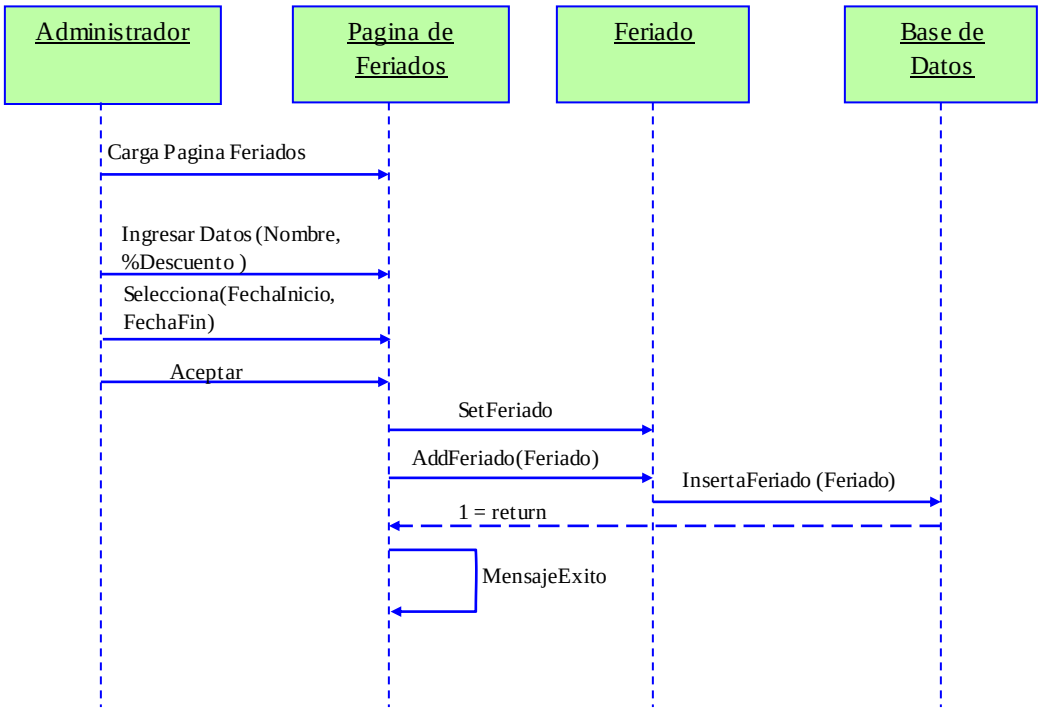


Figura 31. Ingreso de feriados.

Consulta Habitación.

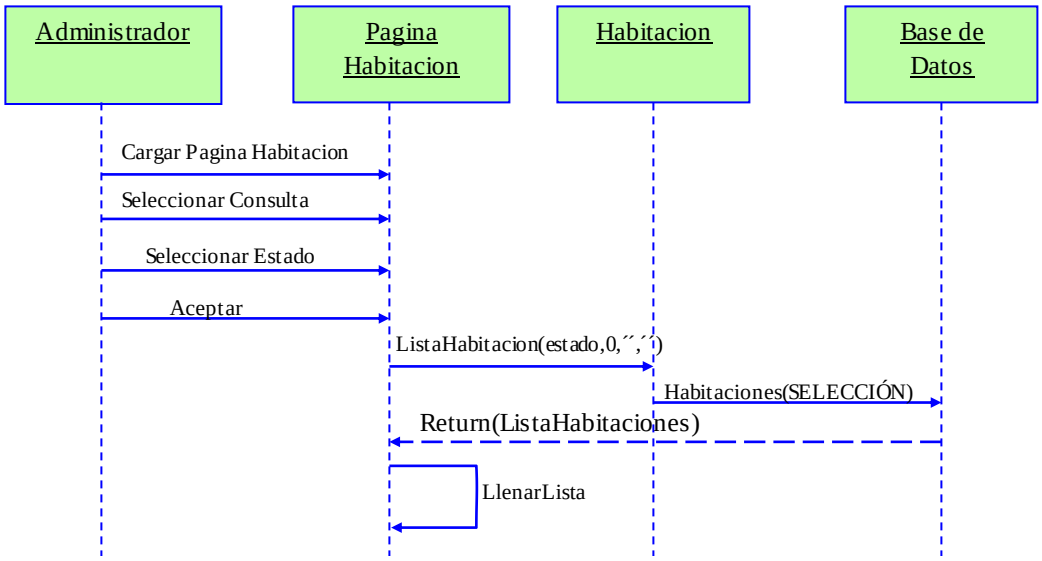


Figura 32. Consulta de habitaciones.

3.6 Modelo de Objetos de Diseño

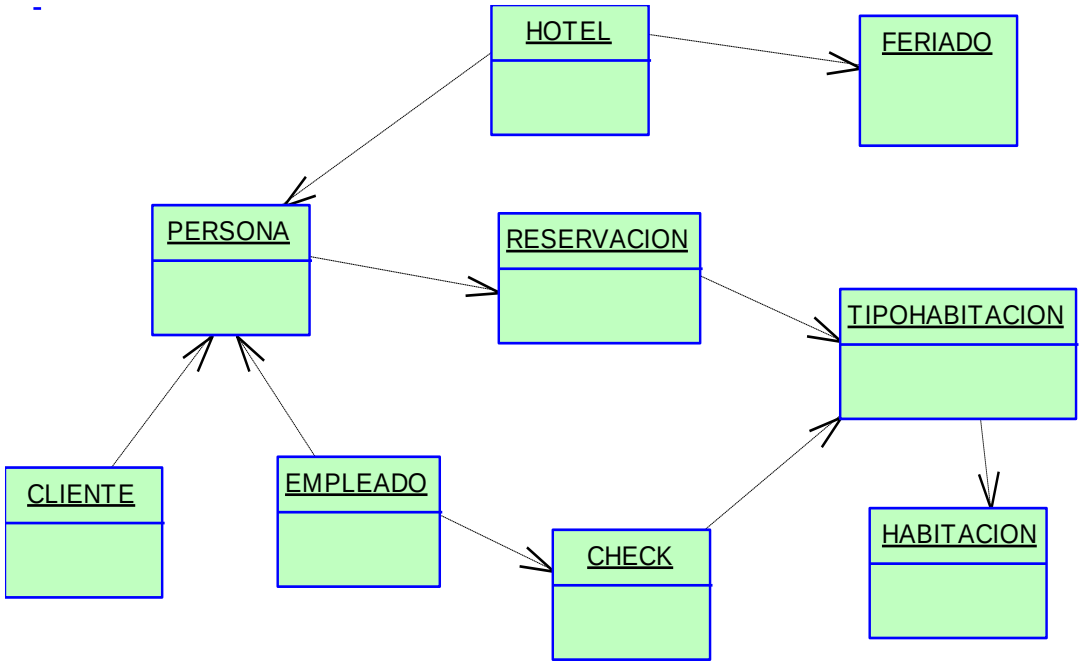


Figura 33. Diagrama de Objetos.

3.7 Diagrama de Clases

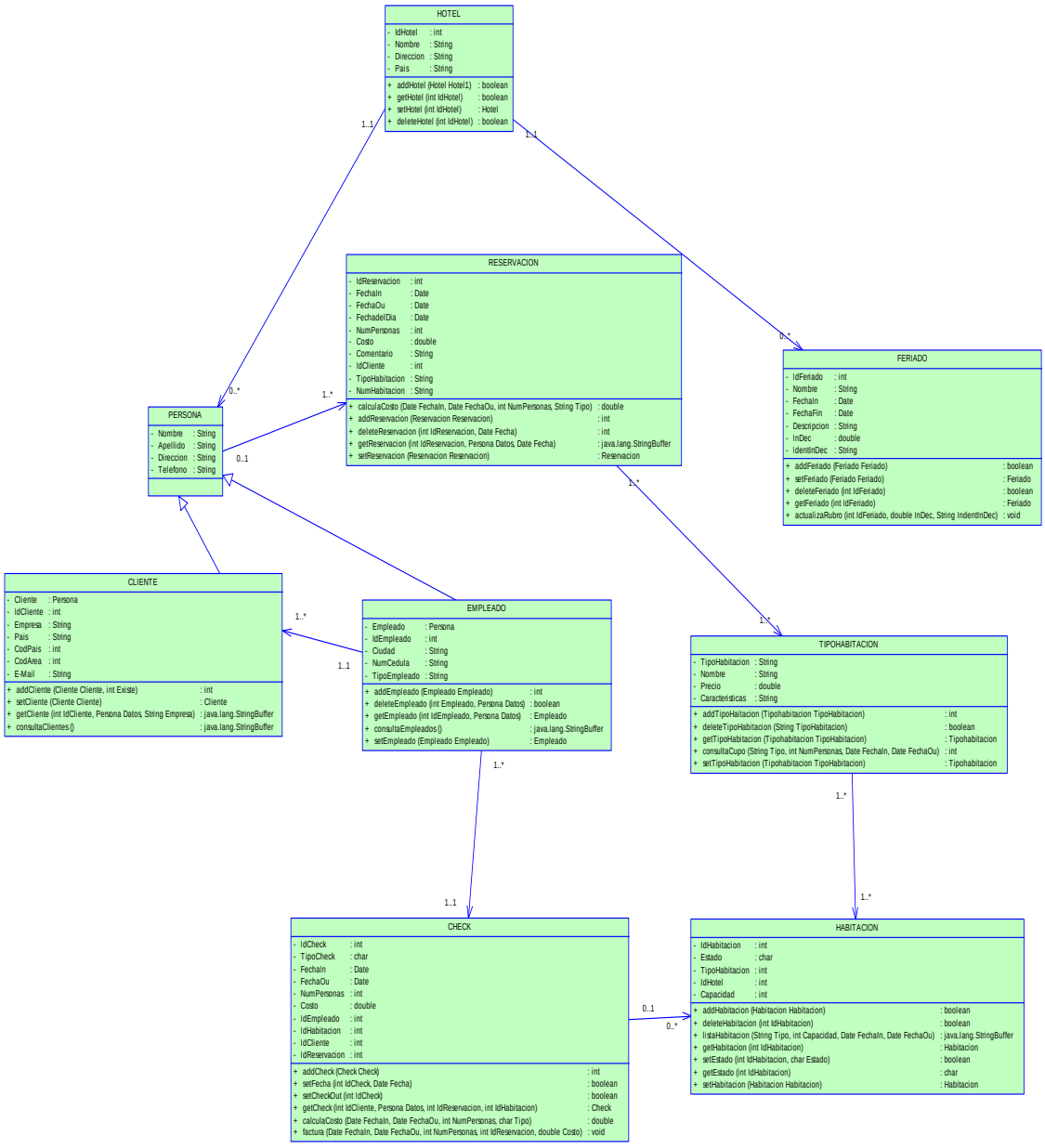


Figura 34. Diagrama de Clases.

3.8 Flujo de ventanas

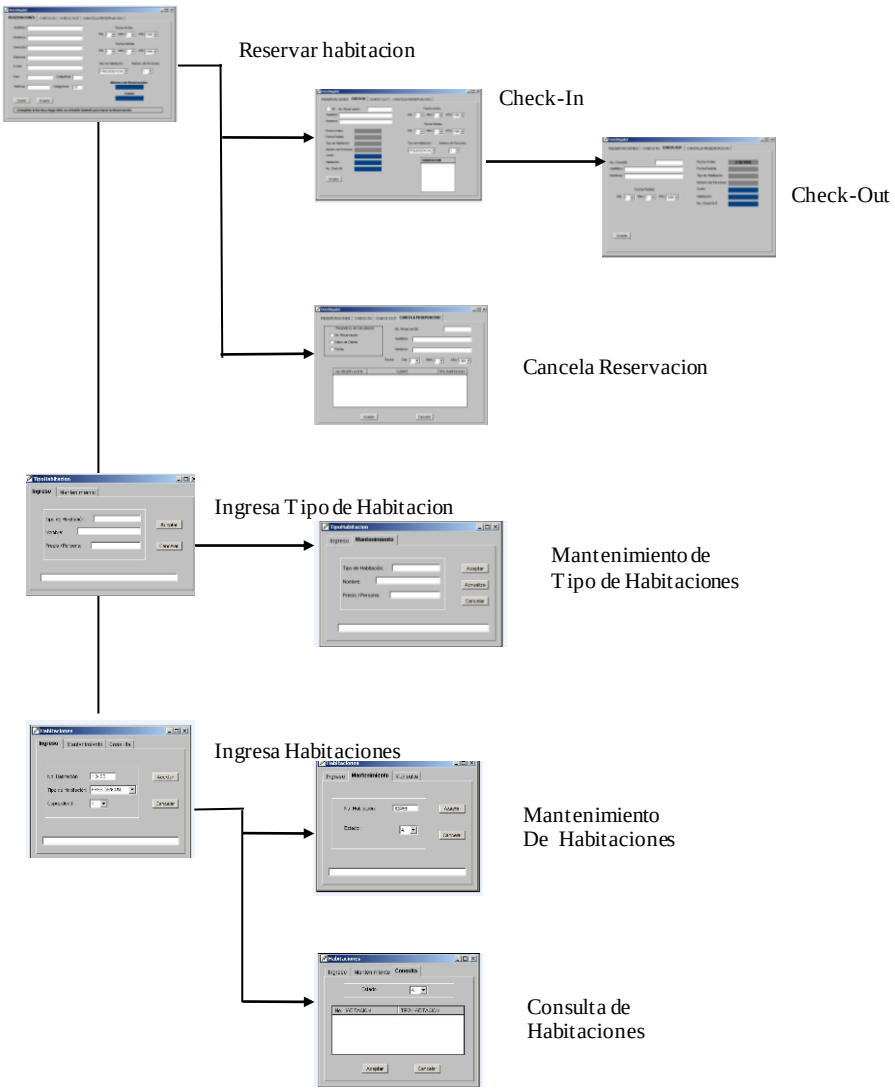


Figura 35. Flujo de Ventanas.

3.9 Layouts de ventanas

Para una mejor visualización de las ventanas ver el manual del sistema en el Anexo 1

3.10 Plan de pruebas

Nombre del Grupo de prueba: Ingreso de Reservación

Numero de prueba:	**1.1**
Prerrequisitos:	Julio Cevallos desea hacer una reservación de una habitación del tipo 3 para las fechas del 6 de agosto de 2005 al 12 de agosto de 2005 para dos personas.
Instrucciones de Configuración:	El sistema de reservación tiene registradas 9 habitaciones de tipo 3, con capacidad para dos personas, disponibles en las fechas del 6 al 12 de agosto de 2005. El sistema tiene registrado el día 10 de agosto de 2005, como el feriado Independencia. EL sistema tiene registrado el costo diario por persona para las habitaciones del tipo 3 de 80 dólares. El costo total por la habitación del tipo 3 para dos

	personas durante 7 días es de: 1132.8 dólares.
Instrucciones de Prueba:	Julio Cevallos ingresa sus datos personales correctamente. Julio ingresa la información solicitada para su reservación correctamente Julio está de acuerdo con el monto de reservación y acepta la reservación.
Comportamiento aceptable: El sistema ingresa a la base de datos los datos personales de Julio Cevallos. El sistema ingresa a la base de datos los datos de la reservación El sistema genera un número de reservación para Julio Cevallos	

Nombre del Grupo de prueba: Cancela una Reservación

Numero de prueba:	**1.2**
Prerrequisitos:	El recepcionista desea eliminar una reservación con numero 99067. La fecha en que se encuentra el sistema es 15 de agosto de 2005.
Instrucciones de Configuración:	El sistema tiene registrada la reservación con número 99067. La reservación con número 99067 tiene fecha de check-in 14 de agosto de 2005.
Instrucciones de Prueba:	Llegada la fecha de check-in y el cliente no se ha acercado ha realizar el check-in de la reservación, se debe proceder a eliminar dicha reservación. El recepcionista confirma que la reservación debe ser eliminada por tener atrasado el check-in.
omportamiento aceptable:	
El sistema elimina de la base de datos la reservación con número 99067	

Nombre del Grupo de prueba: Check-In de una habitación con reservación.

Numero de prueba:	**2.1**
Prerrequisitos:	Julio Cevallos solicita el check-in de una habitación con número de reservación 99078 el 6 de agosto de 2005.
Instrucciones de Configuración:	El sistema tiene 200 reservaciones, incluyendo la reservación a nombre de Julio Cevallos con el número 99078. El sistema tiene los datos personales de Julio. EL sistema tiene las siguientes habitaciones de tipo 3 disponibles: 201A,306A,307A,115B,116B.
Instrucciones de Prueba:	El recepcionista pide la confirmación a Julio Cevallos para registrar el check-in en base a la información que se encuentra en la reservación con número 99078 El recepcionista selecciona la habitación 306A y acepta el check-in Julio Cevallos acepta la habitación 306A.

Comportamiento aceptable: El sistema registra la habitación 306A como ocupada. El sistema ingresa a la base de datos la información del check-in.	

Nombre del Grupo de prueba: Check-In de una habitación sin reservación

Numero de prueba:	**2.2**
Prerrequisitos:	Mercedes Sandoval solicita el Check-in de una habitación el día 10 de septiembre de 2005. Mercedes no ha realizado una reservación previamente. Desea utilizar una habitación del tipo Presidencial del 10 al 15 de septiembre de 2005 para una persona.
Instrucciones de Configuración:	El sistema tiene registrado 5 habitaciones simples (capacidad una persona) del tipo Presidencial disponibles para el 10 de septiembre. Las habitaciones son: 103A, 104A, 109A, 204B, 205B. El costo diario por persona para las habitaciones del

	tipo 2 es de 95 dólares. El costo total de la habitación de tipo Presidencial para una persona durante 6 días es de 573.8
Instrucciones de Prueba:	Mercedes acepta el check-in y el monto que debe cancelar al hacer el check-out. El recepcionista ingresa correctamente los datos personales de Mercedes. El recepcionista escoge la habitación 103A y acepta el check-in de Mercedes Se entrega la habitación 103A a Mercedes.
Comportamiento aceptable: El sistema actualiza la habitación 103A como ocupada. EL sistema ingresa en la base de datos los datos personales de Mercedes El sistema ingresa la información del check-in.	

Nombre del Grupo de prueba: Check-Out

Numero de prueba:	**2.3**
Prerrequisitos:	Julio pide el check-out de la habitación 306A el día 9 de agosto de 2005, y solicita la cuenta.
Instrucciones de Configuración:	El sistema tiene registrado el check-in número 99066 a nombre de Julio con fecha de check-in de agosto de 2005, la habitación 306A por los días del 4 al 9 de agosto de 2005. El sistema tiene registrado el costo diario por persona para las habitaciones de l tipo 3 de 80 dólares. El monto total a pagar por la habitación durante 4 días es 326.4 dólares.
Instrucciones de Prueba:	El recepcionista le comunica a Julio, que el monto a cancelar es: 326.4 dólares, Julio cancela la cuenta.
Comportamiento aceptable:	

El sistema actualiza en la base de datos la habitación 306A como disponible El sistema registra en la base de datos la información del check-out de la habitación 306A.

Nombre del Grupo de prueba: Ingreso de habitación

Numero de prueba:	**2.4**
Prerrequisitos:	El administrador desea registrar en el sistema una nueva habitación, que ha sido adecuada para su uso. Desea registrar la habitación con numero 320B, que pertenezca al tipo de habitación Normal y con capacidad para dos personas. Características: cama doble, televisor.
Instrucciones de Configuración:	El sistema no tiene registrada una habitación con el número 320B. Sistema tiene registrada el tipo de habitación Normal, con un costo diario por persona de 95 dólares.

Instrucciones de Prueba:	El administrador ingresa correctamente los datos de la nueva habitación • Número de habitación: 320B • Tipo de habitación: NORMAL • Capacidad: 2 • Características: cama doble, televisor. El administrador acepta el ingreso.
Comportamiento aceptable: El sistema ingresa en la base de datos, una habitación con número de habitación 320B, con las características especificadas, y la registra como disponible.	

Nombre del Grupo de prueba: Ingreso de Tipo de habitación

Numero de prueba:	**2.5**
Prerrequisitos:	El administrador desea registrar un nuevo tipo de habitación. Quiere especificar un precio específico para algunas habitaciones que poseen jacuzzi.

	Estas habitaciones desea registrarlas con el nuevo tipo de habitación. Las habitaciones que pertenezcan a este tipo tendrán un costo diario por persona de 120 dólares por personas.
Instrucciones de Configuración:	El sistema tiene registrado 4 tipos de habitaciones
Instrucciones de Prueba:	El administrador ingresa los datos de tipo de habitación correctamente. Especifica las características, lo destacable de las habitaciones de este tipo que justifique su precio.
Comportamiento aceptable: El sistema ingresa en la base de datos el nuevo tipo de habitación.	

Nombre del Grupo de prueba: Ingreso de Feriado

Numero de prueba:	**2.6**
Prerrequisitos:	El administrador desea especificar la fecha del 25 de diciembre como un feriado con nombre Navidad. Desea

	especificar que el feriado tendrá un porcentaje de recargo de 5% sobre el precio diario por persona de las habitaciones.
Instrucciones de Configuración:	En el sistema no se encuentra definido ningún feriado que contenga el 25 de diciembre.
Instrucciones de Prueba:	El administrador ingresa correctamente la información del feriado Navidad. El administrador acepta el ingreso del nuevo feriado.
Comportamiento aceptable: El sistema ingresa en la base de datos el nuevo feriado.	

3.11 Diseño de Base de Datos

3.11.1 Descripción de Tablas Principales

CLIENTE

Contiene los datos, que indican la información para identificar a un cliente.

- **IDCLIENTE** **INT (4)** **NOT NULL** **PK**
- **IDHOTEL** **TINYINT** **NOT NULL** **PK**
- **IDEMPLEADO** **INT (4)** **NOT NULL** **PK**
- NOMBRE VARCHAR (50) NOT NULL
- APELLIDO VARCHAR (50) NOT NULL
- EMPRESA VARCHAR (50)
- DIRECCION VARCHAR (50) NOT NULL
- PAIS VARCHAR (50) NOT NULL
- CODPAIS VARCHAR (5) NOT NULL
- CODAREA VARCHAR (5) NOT NULL
- TELEFONO VARCHAR (20) NOT NULL
- EMAIL VARCHAR (50)

CLIENTE - Especificación de campos.

IDCLIENTE	Código del cliente.
IDHOTEL	Código del hotel donde se realiza reservación/ hospedaje el cliente
IDEMPLEADO	Código del Empleado
NOMBRE	Nombres del cliente.
APELLIDO	Apellidos del cliente.
EMPRESA	Empresa en la que labora el cliente.
DIRECCION	Dirección de domicilio o de la empresa del cliente.
PAIS	País de donde es originario el cliente.
CODPAIS	Código del país del cual procede el cliente.
CODAREA	Código de la región de la que procede el cliente.
TELEFONO	Numero Telefónico, donde se puede contactar al cliente.
EMAIL	Dirección de correo electrónico, donde se puede contactar al cliente.

HOTEL

Contiene los datos, que indican la información para identificar a un hotel.

- **IDHOTEL** **TINYINT** **NOT NULL** **PK**

- NOMBRE VARCHAR (50) NOT NULL
- DIRECCION VARCHAR (50) NOT NULL
- CIUDAD VARCHAR (50) NOT NULL
- PAIS VARCHAR (50) NOT NULL

HOTEL - Especificación de campos.

IDHOTEL Código del hotel.

NOMBRE Nombre que identifica al hotel.
DIRECCION Dirección de donde se encuentra ubicado el hotel.
CIUDAD Ciudad en la que se encuentra el hotel.
PAIS País en el que se encuentra el hotel.

RESERVACION

Contiene los datos, que indican la información, de reservaciones por cliente.

- **IDRESERVACION INT (4) NOT NULL PK**
- **IDCLIENTE INT (4) NOT NULL PK**
- **TIPOHABITACION CHAR (2) NOT NULL PK**
- FECHAIN DATETIME (8) NOT NULL
- FECHAOU DATETIME (8) NOT NULL
- NUMPERSONAS TINYINT (1) NOT NULL
- COSTO INT (4) NOT NULL
- COMENTARIO VARCHAR (50)
- FECHADELDIA DATETIME (8) NOT NULL (dateadd(day,0,getdate()))
- ESTADO CHAR (2) NOT NULL ('A')

RESERVACION - Especificación de campos.

IDRESERVACION Código de la reservación, realizada por un cliente o empleado.

IDCLIENTE Código del cliente.
TIPOHABITACION Código del tipo de habitación, de acuerdo a ciertas características similares.
FECHAIN Fecha de check-in, fecha en la que el cliente se va ha presentar en el hotel.
FECHAOU Fecha de check-out, fecha en la que el cliente termina su estadía en el hotel, fecha de salida.
NUMPERSONAS Número de personas que se hospedaran en el hotel.
COSTO El valor que deberá cancelar el cliente, de acuerdo al tipo de

	habitación, tiempo de estadía y número de personas.
COMENTARIO	Texto adicional para agregar mayor información.
FECHADELDIA	La fecha actual.
ESTADO	Indica si esta en proceso (‘A’), si se ha hecho check-In (‘I’), o se ha realizado check-out (‘C’).

CHECK

Contiene los datos, que indican la información, de check-in/check-out, por cada cliente.

–	**IDCHECK**	**INT (4)**	**NOT NULL**	**PK**
–	**IDRESERVACION**	**INT (4)**	**NOT NULL**	**PK**
–	**IDCLIENTE**	**INT (4)**	**NOT NULL**	**PK**
–	**IDEMPLEADO**	**INT (4)**	**NOT NULL**	**PK**
–	**NUMHABITACION**	**VARCHAR (5)**	**NOT NULL**	**PK**
–	TIPOCHECK	CHAR (2)	NOT NULL	
–	FECHAIN	DATETIME (8)	NOT NULL	
–	FECHAOU	DATETIME (8)	NOT NULL	
–	NUMPERSONAS	TINYINT	NOT NULL	
–	COSTO	INT (4)	NOT NULL	

CHECK - Especificación de campos.

IDCHECK	Código del check, check-in o check-out.
IDRESERVACION	Código de la reservación.
IDCLIENTE	Código del cliente.
IDEMPLEADO	Código del empleado, que realiza en check.
NUMHABITACION	Él número de habitación que se le asigna a un cliente.
TIPOCHECK	Identifica si se está realizando un check-in (‘I’), o un check-out (‘O’).
FECHAIN	La fecha en que el cliente ingresa a la habitación.
FECHAOU	La fecha en la que el cliente, entrega la habitación y cancela.
NUMPERSONAS	El número de personas hospedadas en la habitación.
COSTO	El costo a cancelar por el cliente, de acuerdo al tiempo de hospedaje, número de personas.

TIPOHABITACION

Contiene los datos, que describen la información almacenada para indicar categorías de habitaciones.

–	**TIPOHABITACION**	**CHAR (2)**	**NOT NULL**	**PK**
–	NOMBRE	VARCHAR (50)	NOT NULL	
–	PRECIO	INT (4)	NOT NULL	

TIPOHABITACION - Especificación de campos.

TIPOHABITACION	El identificador para categorizar los distintos tipos de habitación, de acuerdo a las características de las habitaciones.
NOMBRE	Nombre del tipo de habitación
PRECIO	Costo de la habitación de acuerdo al tipo.

HABITACION

Contiene los datos, que describen los datos que contiene una habitación.

–	**NUMHABITACION**	**VARCHAR (5)**	**NOT NULL**	**PK**
–	**TIPOHABITACION**	**CHAR (2)**	**NOT NULL**	**PK**
–	**IDHOTEL**	**TINYINT**	**NOT NULL**	**PK**
–	CAPACIDAD	TINYINT	NOT NULL	
–	ESTADO	CHAR (2)	NOT NULL	

HABITACION - Especificación de campos.

NUMHABITACION	El código que identifica a la habitación.
TIPOHABITACION	La categoría a la que pertenece la habitación, de acuerdo a las características de esta.
IDHOTEL	El hotel donde se encuentra asignada esta habitación.
CAPACIDAD	La cantidad de personas que pueden hospedarse en esta habitación.
ESTADO	Indica si la habitación, está disponible o no.

FERIADO

Contiene los datos, que indican la información, que describe a un feriado.

–	**IDFERIADO**	**INT (4)**	**NOT NULL**	**PK**
–	**IDHOTEL**	**TINYINT**	**NOT NULL**	**PK**
–	NOMBRE	VARCHAR (50)	NOT NULL	
–	FECHAIN	DATETIME (8)	NOT NULL	
–	FECHAFIN	DATETIME (8)	NOT NULL	

- DESCRIPCION TEXT (16)
- INDEC TINYINT NOT NULL
- IDENTINDEC CHAR (2) NOT NULL

FERIADO - Especificación de campos.

IDFERIADO	Código que identifica al feriado.
IDHOTEL	Código que identifica al hotel donde se define el feriado.
NOMBRE	Nombre del feriado
FECHAIN	Fecha de inicio del feriado.
FECHAFIN	Fecha de culminación del feriado.
DESCRIPCION	Una descripción detallada del feriado.
INDEC	Porcentaje de incremento o decremento al costo de la habitación, para esta fecha de feriado.
IDENTINDEC	Signo '+' o '-' para indicar sí se trata de un incremento o un decremento al costo de la habitación para este feriado.

3.11.2 Diagrama Entidad Relación

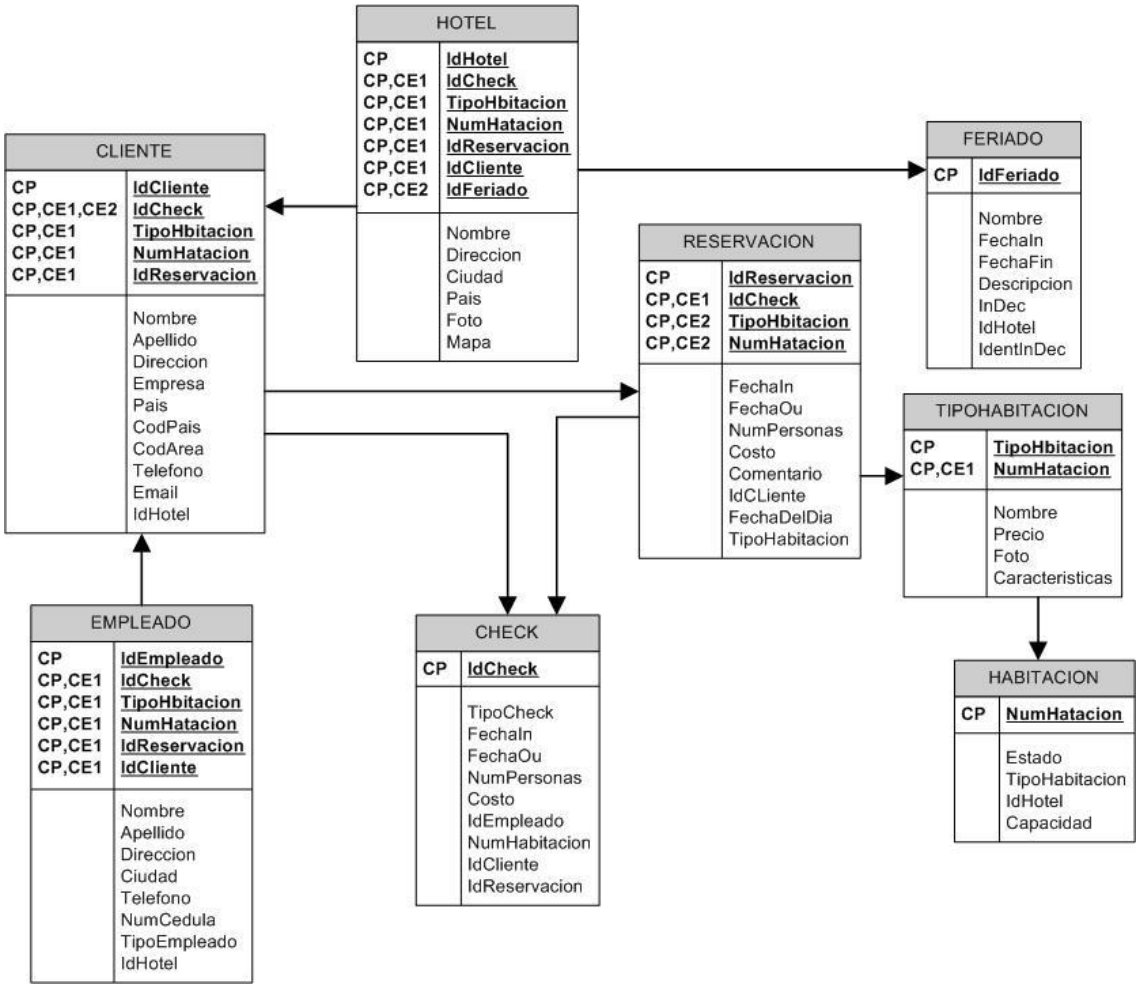


Figura 36. Diagrama Entidad Relación.

3.11.3 Descripción de los Procedimientos almacenados

Procedimientos Almacenados Importantes

IngresaCliente

Este proceso crea el registro en la base de datos, ya sea por el cliente al ingresar reservación o por el empleado al ingresar check-in.

ENTRADA:

Nombre:	Nombres del cliente.
Apellido:	Apellidos del cliente.
Empresa:	Empresa en la que labora el cliente.
Direccion:	Dirección, donde se puede localizar al cliente.
Pais:	País de residencia del cliente.
CodPAis:	Código del país donde reside el cliente.
CodArea:	Código de la ciudad donde reside el cliente.
Telefono:	El teléfono donde se pude contactar al cliente.
Email:	Dirección de correo electrónico, donde se puede contactar al cliente.
IdHotel:	Código del hotel donde el cliente hace una reservación o check-in.
IdEmpleado:	Código del empleado que atiende el requerimiento del cliente.

SALIDA:

IdCliente:	Indica el código del cliente ingresado, si no se asigna uno, se retorna mensaje de error.
Existe:	Indica el status del cliente. Si existe retorna un 1, sino un 0.

```
CREATE PROCEDURE IngresaCliente  @Nombre varchar(50)      @Apellido varchar(50),
                                 @Empresa varchar(50),    @Direccion varchar(50),
                                 @Pais varchar(50),       @CodPais varchar(5),
                                 @CodArea varchar(4),     @Telefono varchar(20),
                                 @Email varchar (50),     @IdHotel tinyint,
                                 @IdEmpleado int,
                                 @IdCliente int OUTPUT,
                                 @Existe int OUTPUT
```

IngresaReservacion

Este proceso crea el registro en la base de datos, ya sea por el cliente al realizar una reservación, o por el empleado del hotel, al recibir una llamada de un posible cliente.

ENTRADA:

FechaIn:	Fecha en la que el cliente, tiene pensado hospedarse en el hotel.
FechaOu:	Fecha en la que el cliente, tiene pensado desalojar el hotel.
NumPersonas:	Cantidad de personas, a hospedarse.
Costo:	El costo, a facturarse al cliente, de acuerdo a las fechas de hospedaje, tipo de habitación y numero de personas.
IdCliente:	Código del cliente que realiza, o solicita la reservación.
TipoHabitacion:	La categoría de la habitación, seleccionada.

SALIDA:

IdReservacion:	Retorna el código e la reservación realizada.

```
CREATE PROCEDURE IngresaReservacion @FechaIn datetime , @FechaOu datetime,
@NumPersonas tinyint , @Costo int,
@IdCliente int, @TipoHabitacion char(2),
@IdReservacion int OUTPUT
```

IngresaCheckIn

Este proceso crea el registro en la base de datos, el check-in, datos de hospedaje en el hotel por un cliente e ingresado por un empleado.

ENTRADA:

FechaIn:	Fecha en la que el cliente, ingresa al hotel.
FechaOu:	Fecha en la que el cliente, tiene pensado desalojar el hotel.
NumPersonas:	Cantidad de personas, a hospedarse.
IdCliente:	Código del cliente, al que se le realiza el check-in.
IdReservacion:	De existir, se ingresa los datos con esta reservación.
TipoCheck:	Estatus del check, en este caso check-in ('Ì').
Habitacion:	Numero de la habitación asignada al cliente.

SALIDA:

IdCheck:	Retorna el código del check-in.

```
CREATE PROCEDURE IngresaCheckIn @FechaIn datetime , @FechaOu datetime,
@NumPersonas tinyint , @Costo int,
@IdCliente int, @IdReservacion int,
@TipoCheck char, @Habitacion varchar(10),
@IdCheck int OUTPUT
```

IngresaCheckOut

Este proceso crea el registro en la base de datos, el check-out, datos de salida del cliente en el hotel.

ENTRADA:

IdCliente:	Código del cliente que esta hospedado.
IdCheck:	Código del check-in, al que se realiza check-out
Costo:	Valor a cancelar por el cliente.
FechaOu:	La fecha en la que el cliente solicita el check-out, fecha en la que el cliente abandona el hotel.

SALIDA:

Existe:	Variable de status, retorna un 1 si existe el check-in, y un 0 de no existir, y retorna un mensaje de error.
NumFactura:	Código de la factura que contiene el detalle de los datos de check-in y check-out del cliente.

```
CREATE PROCEDURE IngresaCheckOut   @IdCliente int, @IdCheck int, @Costo int,
                                   @FechaOu datetime,
                                   @Existe int OUTPUT,
                                   @NumFactura int OUTPUT
```

CalculaCosto

Este proceso calcula el costo (valor a cancelar), por el cliente de acuerdo al tiempo de hospedaje, numero de personas, y el tipo de habitación.

ENTRADA:

FechaIn:	Fecha de check-in, de registro del cliente al hotel.
FechaOu:	Fecha de check-out, de salida del cliente del hotel.
NumPersonas:	El número de personas hospedadas en la habitación.
TipoHabitacion:	La categoría de la habitación.

SALIDA:

Costo: El valor a cancelar por el cliente, por el hospedaje.

CREATE PROCEDURE CalculaCosto @FechaIn datetime, @FechaOu datetime, @NumPersonas tinyint, @TipoHabitacion char, @Costo int OUTPUT

ExtHabDisp

Este proceso retorna una lista de habitaciones disponibles, de acuerdo a un rango de fechas, tipo de habitación y número de personas.

ENTRADA:

FechaIn: Fecha del posible check-in, de registro del cliente al hotel.
FechaOu: Fecha del posible check-out, de salida del cliente del hotel.
NumPersonas: Él número de personas que se hospedaran en la habitación.
TipoHabitacion: La categoría de la habitación.

SALIDA:

Existe: Variable de status, si existe habitación/es disponibles, se retorna un 1, si no existe por lo menos una habitación disponible se retorna un 0, y un mensaje de error.

CREATE PROCEDURE ExtHabDisp @FechaIn datetime, @FechaOu datetime,

@TipoHabitacion char, @NumPersonas tinyint, @Existe int OUTPUT

CAPÍTULO 4

4 IMPLEMENTACIÓN DEL PROYECTO

4.1 Introducción

En este capítulo se define la estructura de cada uno de los componentes clientes, servidores y conexión a la base de datos, en sus partes más importantes como son el IDL, la inicialización de los ORB y de BOA, y los métodos en java que hacen llamadas a procedimientos almacenados en la base de datos.

4.2 Proceso Cliente

Struct del IDL

```
// Cliente.idl

module Hotel
{

  struct Persona
  {
    string  Nombre ;
    string  Apellido ;
    string  Direccion ;
    string  Telefono ;
  };

  struct EmpleadoStruct
  {
    Persona Empleado ;
    long    IdEmpleado ;
    string  Ciudad ;
    string  NumCedula ;
    string  TipoEmpleado ;
  };
  typedef sequence<EmpleadoStruct> EmpleadoSeq;

 struct ClienteStruct
  {
    Persona Cliente ;
    long    IdCliente ;
    string  Empresa ;
    string  Pais ;
    string  CodPais ;
    string  CodArea ;
    string  Email ;
  };
  typedef sequence<ClienteStruct> ClienteSeq;
```

4.3 Proceso Servidor

Interfases del IDL

```
interface Cliente
{
   //Devuelve el IdCliente y si existe, lo añade cuando no esxiste
   long AddCliente ( in ClienteStruct Cliente, out long Existe);

   //Retorna un cliente o grupo de clientes con sus datos
   ClienteSeq GetCliente (in long IdCliente, in Persona Datos, in
string Empresa );
   // Retorna toda una la lista de Clientes Futuros
   ClienteSeq ConsultaClientes ();

   //Setea o actualizo el Cliente
   ClienteStruct SetCliente (in ClienteStruct Cliente );

};



interface Empleado
{

    //Se ingresa un nuevo Empleado
    long AddEmpleado ( in EmpleadoStruct Empleado);

    //Borra un Empleado
    boolean DeleteEmpleado ( in long IdEmpleado, in Persona Datos );

    //Retorna informacion del empleado
    EmpleadoStruct GetEmpleado (in long IdEmpleado, in Persona Datos);

    //Retorna todos los empleados FUTURO
    EmpleadoSeq ConsultaEmpleados ();

    //Seteo o actualizo A un Empleado
    EmpleadoStruct SetEmpleado (in EmpleadoStruct Empleado);

};
```

Clase Servidora

```
//package ServidorHotel;

//import ProyectoHotel.Hotel.*;
class Servidor
{
 static public void main(String[] args)
  {
   try
    {
      // Inicializa el ORB
      System.out.println("Inicializando el ORB");
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);

      // Inicializa el BOA
      System.out.println("Inicializando Boa");
      org.omg.CORBA.BOA boa = orb.BOA_init();

      // Crea el objeto Cliente
      System.out.println("Crea Cliente");
      ClienteImp Cliente = new ClienteImp ("Cliente");

      // Crea el objeto Empleado
      System.out.println("Crea Empleado");
      EmpleadoImp Empleado = new EmpleadoImp ("Empleado");

.
.
.


      // Exporta al ORB los objetos
      boa.obj_is_ready(Cliente);
      boa.obj_is_ready(Empleado);
.
.
.

      // Listo para atender requerimientos
      System.out.println("Listo Para Atender");
      boa.impl_is_ready();
   }catch(org.omg.CORBA.SystemException e)
      {
        System.err.println(e);
      }
     System.out.println("despues del Catch del orb");
  }
```

```
}
```

Clase que conecta al Servidor con la Base de Datos

```
//package ServidorHotel;


import java.net.URL;
import java.sql.*;
import java.util.*;

public class HotelDB
{
  Connection con;
  Driver driver = null;
  ResultSet rs;
  PreparedStatement pstmt;
  CallableStatement sp ;

  public void conexion() throws Exception
  {
    try
    {
      // Load the jdbc-odbc bridge driver
      Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");

      String url   = "jdbc:odbc:hotel" ;
      System.out.println("Conectando a " + url);
      con = DriverManager.getConnection(url, "proyecto", "topico");
    } catch(Exception e)
    {
      System.err.println( "System Exception in conexion");
      System.err.println(e);
      throw e;
    }
  }


  public void Close() throws Exception
  {
    try
    {
      System.out.println("Closing connection");
      con.close();
    } catch (Exception e)
    {
      System.err.println("System Exception in close");
      System.err.println(e);
```

```
      throw e;
    }
  }

.
.
.
public int InsertaCliente (Hotel.ClienteStruct Cliente ) throws
Exception
{
 int IdCliente, Existe ;
    try
    {

      sp=con.prepareCall("{call
IngresaCliente(?,?,?,?,?,?,?,?,?,?,?,?,?)}");
      sp.registerOutParameter(12,java.sql.Types.INTEGER);
      sp.registerOutParameter(13,java.sql.Types.INTEGER);
      sp.setString(1,Cliente.Cliente.Nombre);
      sp.setString(2,Cliente.Cliente.Apellido);
      sp.setString(3,Cliente.Empresa);
      sp.setString(4,Cliente.Cliente.Direccion);
      sp.setString(5,Cliente.Pais);
      sp.setString(6,Cliente.CodPais);
      sp.setString(7,Cliente.CodArea);
      sp.setString(8,Cliente.Cliente.Telefono);
      sp.setString(9,Cliente.Email);
      sp.setInt(10,1);
      sp.setInt(11,10);
      int RetVal=sp.executeUpdate();
      System.out.print("IngresaCliente ");
      System.out.println(RetVal);
      IdCliente = sp.getInt(12);
      Cliente.IdCliente=IdCliente ;
      Existe=sp.getInt(13);
      sp.close();
      System.out.println(Cliente.IdCliente);
    } catch( Exception e ){
      System.err.println("System Exception in InsertaCliente");
      System.err.println("Error en HotelDB");
      System.err.println(e);
      throw e;
    }

  return IdCliente ;

}
```

4.4 Procesos de la Base de Datos

4.4 1 Procesos Almacenados de la Base de Datos

```
CREATE PROCEDURE IngresaCheckIn  @FechaIn datetime , @FechaOu datetime,
@NumPersonas tinyint , @Costo int,
@IdCliente int,  @IdReservacion int,@TipoCheck char, @Habitacion
varchar(10), @IdCheck int OUTPUT
AS

INSERT INTO [CHECK]
(TipoCheck,FechaIn, FechaOu, NumPersonas, Costo,
IdEmpleado,NumHabitacion,IdCliente, IdReservacion)
VALUES
('I',@FechaIn, @FechaOu, @NumPersonas, @Costo,
10,@Habitacion,@IdCliente, @IdReservacion)

SELECT @IdCheck = IdCheck FROM [CHECK] WHERE IdCliente = @IdCliente AND
Costo = @Costo OR IdReservacion = @IdReservacion
INSERT INTO PRUEBA (Valor,Descripcion) VALUES (@IdReservacion,'CHECK')
RETURN @IdCheck



CREATE PROCEDURE IngresaCheckOut  @IdCliente int, @IdCheck int, @Costo
int, @FechaOu datetime,
@Existe int OUTPUT, @NumFactura int OUTPUT
AS

 IF EXISTS (SELECT IdCheck FROM [CHECK] WHERE IdCheck= @IdCheck OR
IdCliente = @IdCliente)
         BEGIN
              UPDATE [CHECK] SET TipoCheck = 'O',FechaOu = @FechaOu,
Costo = @Costo WHERE IdCheck= @IdCheck
              SET  @Existe =1
              SELECT @NumFactura = NumFactura FROM FACTURA  WHERE Dato
= '0'
       SET @NumFactura = @NumFactura + 1
              UPDATE FACTURA  SET NumFactura = @NumFactura WHERE Dato =
'0'
```

```
      RETURN @NumFactura
         END
      ELSE
         BEGIN
                  SET @Existe =0
              RETURN @Existe
       END



CREATE PROCEDURE IngresaCliente @Nombre varchar(50), @Apellido
varchar(50), @Empresa varchar(50),
@Direccion varchar(50), @Pais varchar(50), @CodPais varchar(5),
@CodArea varchar(4), @Telefono varchar(20),
@Email varchar (50), @IdHotel tinyint, @IdEmpleado int, @IdCliente int
OUTPUT,
@Existe int OUTPUT
 AS
     IF EXISTS (SELECT IdCliente FROM CLIENTE WHERE Nombre = @Nombre
AND Apellido = @Apellido)
          BEGIN
              SET @Existe =1
          END
     ELSE
          BEGIN
              INSERT INTO CLIENTE
              (Nombre, Apellido, Empresa, Direccion, Pais, CodPais,
CodArea, Telefono, Email, IdHotel, IdEmpleado)
              VALUES
              (@Nombre, @Apellido, @Empresa, @Direccion, @Pais,
@CodPais, @CodArea, @Telefono, @Email, @IdHotel, @IdEmpleado)
              SET  @Existe =0
          END

        SELECT @IdCliente = IdCliente FROM CLIENTE WHERE Nombre =
@Nombre AND Apellido = @Apellido
        insert into prueba (Valor,Descripcion)
values(@IdCliente,'IDCLIENTE')
        RETURN @IdCliente



CREATE PROCEDURE IngresaReservacion @FechaIn datetime , @FechaOu
datetime, @NumPersonas tinyint , @Costo int,
@IdCliente int, @TipoHabitacion char(2), @IdReservacion int OUTPUT
 AS

INSERT INTO RESERVACION
```

```
(FechaIn, FechaOu, NumPersonas, Costo, IdCliente, TipoHabitacion)
VALUES
(@FechaIn, @FechaOu, @NumPersonas, @Costo, @IdCliente, @TipoHabitacion)

SELECT @IdReservacion = IdReservacion FROM RESERVACION WHERE IdCliente
= @IdCliente AND Costo = @Costo
INSERT INTO PRUEBA (Valor,Descripcion) VALUES
(@IdReservacion,'IDRESERVACION')
RETURN @IdReservacion

CREATE PROCEDURE CalculaCosto @FechaIn datetime, @FechaOu  datetime,
@NumPersonas tinyint,
@TipoHabitacion char, @Costo int OUTPUT
AS

DECLARE @Costo1 int
DECLARE @Costo2 int
DECLARE @NdiasFer int
DECLARE @CostoTFer int
DECLARE @DiasTFer int
DECLARE @DiasHospedaje int
DECLARE @DiasNoFer int
DECLARE @CostoNoFer int
DECLARE @signo int

SET @Costo1 = 0
SET @Costo2 = 0
SET @NdiasFer = 0
SET @CostoTFer = 0
SET @DiasTFer = 0
SET @DiasHospedaje = 0
SET @DiasNoFer = 0
SET @CostoNoFer = 0
SET @signo = 1

SELECT  @Costo1= Precio FROM TIPOHABITACION WHERE TipoHabitacion =
@TipoHabitacion
SET @Costo1 = @Costo1 * @NumPersonas

DECLARE C1 CURSOR FOR
SELECT  FechaIn, FechaFin, InDec,IdentInDec
FROM  FERIADO

DECLARE @FechaInF datetime
DECLARE @FechaFF datetime
DECLARE @Porcentaje  int
DECLARE @MasMenos char
```

```
OPEN C1
FETCH C1  INTO @FechaInF,@FechaFF,@Porcentaje,@MasMenos
WHILE  @@FETCH_STATUS = 0
       BEGIN
            IF (@FechaIn <= @FechaFF) AND (@FechaOu >= @FechaInF)
                BEGIN
                      IF @MasMenos = '-'
                           SET @signo = -1

                      IF (@FechaInF >= @FechaIn)
                           BEGIN
                                IF (@FechaFF <= @FechaOu)
                                    BEGIN
                                         SET @NdiasFer =
DATEDIFF(dd,@FechaInF,@FechaFF) + 1
                                         SET @Costo2 =
(@signo*((@Costo1 *@Porcentaje)/100)+@Costo1)*@NdiasFer
                                    END
                                ELSE
                                    BEGIN
                                         SET @NdiasFer =
DATEDIFF(dd,@FechaInF,@FechaOu) + 1
                                         SET @Costo2 =
(@signo*((@Costo1 *@Porcentaje)/100)+@Costo1)*@NdiasFer
                                    END
                           END
                      ELSE
                            BEGIN
                                IF (@FechaFF <= @FechaOu)
                                    BEGIN
                                         SET @NdiasFer =
DATEDIFF(dd,@FechaIn,@FechaFF) + 1
                                         SET @Costo2 =
(@signo*((@Costo1 *@Porcentaje)/100)+@Costo1)*@NdiasFer
                                    END
                                ELSE
                                    BEGIN
                                         SET @NdiasFer =
DATEDIFF(dd,@FechaIn,@FechaOu) + 1
                                         SET @Costo2 =
(@signo*((@Costo1 *@Porcentaje)/100)+@Costo1)*@NdiasFer
                                    END
                           END
                END

  INSERT INTO PRUEBA (Valor)
     VALUES (2222222)
             INSERT INTO PRUEBA (Valor)
     VALUES (@Costo2)
```

```
     SET @CostoTFer = @CostoTFer + @Costo2
INSERT INTO PRUEBA (Valor)
   VALUES (@CostoTFer)
     SET @DiasTFer = @DiasTFer  + @NdiasFer
      FETCH C1  INTO @FechaInF,@FechaFF,@Porcentaje,@MasMenos
     END
  DEALLOCATE C1

  SET @DiasHospedaje = DATEDIFF(dd,@FechaIn,@FechaOu) + 1
  SET @DiasNoFer = @DiasHospedaje - @DiasTfer
  SET @CostoNoFer = @DiasNoFer * @Costo1
     INSERT INTO PRUEBA (Valor)
   VALUES (@CostoNoFer)
  SET @Costo = @CostoTFer + @CostoNoFer
   INSERT INTO PRUEBA (Valor)
   VALUES (@Costo)
  RETURN @Costo
```

CAPÍTULO 5

5 IMPLANTACIÓN DEL PROYECTO

5.1 Introducción

A continuación vamos a detallar los requisitos necesarios para la instalación y configuración de cada uno de los ambientes (cliente, servidor, base de datos), las herramientas especiales como Visibroker, para la debida funcionalidad del aplicativo.

5.2 Herramientas Especiales para Instalación del Sistema

La herramienta especial que se utiliza es el Inprise Visibroker. Esta herramienta soporta la tecnología CORBA. Esta herramienta tiene disponible lo siguiente:

- El compilador del IDL a Java
- Inicia el protocolo IIOP que permita la comunicación entre el cliente y el servidor
- La publicación de los esqueletos de las clases del servidor

5.3 Plataforma del sistema

5.3.1 Requisitos del Cliente

Lo importante del lado del cliente es un web browser que soporte la máquina virtual de java. La plataforma y arquitectura pueden ser de cualquier fabricante. Es recomendable una tarjeta de red a 100 Mbps.

5.3.2 Requisitos del Servidor

En el servidor se requiere lo siguiente:

- Arquitectura x86.

- CPU de 733 MHz.
- RAM de 128 MB, recomendable 256 MB.
- 4 GB de espacio libre.
- Plataforma Windows Server 2000.
- Java 2 SDK ver 1.4.2.06.
- Apache Web Server.
- Visibroker ver. 3.4
- Tarjeta de red a 100 MB recomendable.

5.3.3 Requisitos para Base de Datos

En el servidor de base de datos se requiere lo siguiente:

- Arquitectura x86.
- CPU de 733 MHz.
- RAM de 128 MB, recomendable 256 MB.
- 4 GB de espacio libre
- Plataforma Windows Server 2000.
- Microsoft SQL Server 2000.
- Tarjeta de red a 100 MB recomendable.

5.3.4 Requisitos para las Herramientas Especiales

La herramienta especial tiene que soportar lo siguiente

- Visi Broker SmartAgent
- Visi Broker GateKeeper
- Reg-Edit Tool
- BOA (Adaptador Básico de Objetos). Es de gran importancia para las aplicaciones cliente, entre las tareas principales tenemos
    - Registro de objetos con VisiBroker SmartAgent
    - Instalar y registrar el objeto con el repositorio de implementación
    - Almacenar la información del objeto implementado que reside en el servidor.

5.4 Instalación y Configuración de las Herramientas Especiales

Se debe instalar el Inprise Visibroker, seleccionar la ruta de instalación y modificar la variable de entorno CLASSPATH en el sistema operativo.

Luego revisar que el puerto del OSAGENT (1400) esté activo. Para confirmar esto levantamos el Visibroker Reg-Edit Tool y debe aparecer como se muestra en la siguiente figura.

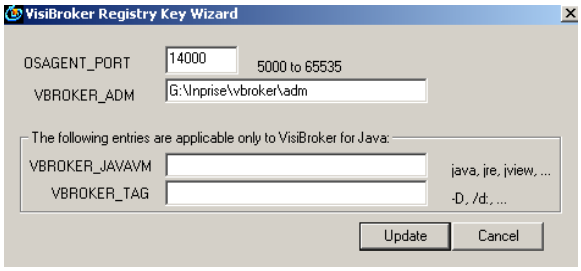


Figura 37: Visibroker Wizard

Se debe crear un archivo de configuración del IOR, a través de la interface Visibroker Configuration, con la cual creamos un archivo gatekeeper.ior. Este archivo debe residir en la ruta donde se encuentra el paquete de la clase servidora.

5.5 Instalación y Configuración del Cliente

En el cliente se debe instalar un web browser. Se puede usar Internet Explorer, Opera, Mozilla, Firefox, Netscape Navigator, etc. Lo importante es que tengan cargada la máquina virtual de java.

5.6 Instalación y Configuración del Servidor

Una vez instalado el sistema operativo (Windows 2000) con el último parche (service pack 4), procedemos a copiar el directorio donde residen nuestros paquetes cliente y servidor. Para esto debemos crear un directorio como se indica a continuación: [driver]:\proyecto\Hotel\, y en esta ubicación colocamos el paquete ProyectoHOTEL.

Luego instalamos el Java 2 SDK, seguimos los pasos de instalación, y revisamos la variable de ambiente PATH (del sistema operativo) para que podamos ejecutar los comandos java y javac desde cualquier ubicación. Si no está definida revisar la ruta de

instalación del JDK en el sistema, y agregarla en la variable PATH (Ej. D:\j2sdk1.4.1_01\bin).

Instalación del Apache Web Server, descargar el paquete appserv-win32-1.8.0.exe, realizar la instalación del mismo, seleccionar instalación personalizada, y marcar la opción Apache, como se muestra en el gráfico

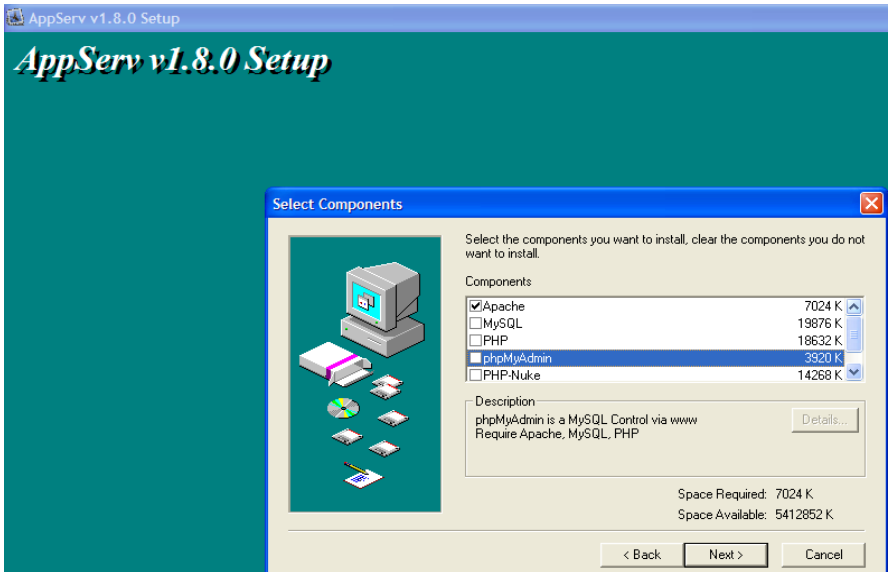


Figura 38. Instalación de Apache Web Server

Indicar el nombre o la dirección IP del host donde estamos instalando el web Server.

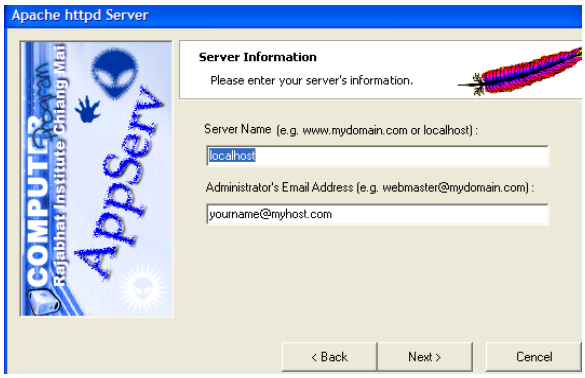


Figura 39. Se indica el nombre del host servidor

Terminada la instalación seleccionamos que se inicie el Apache.

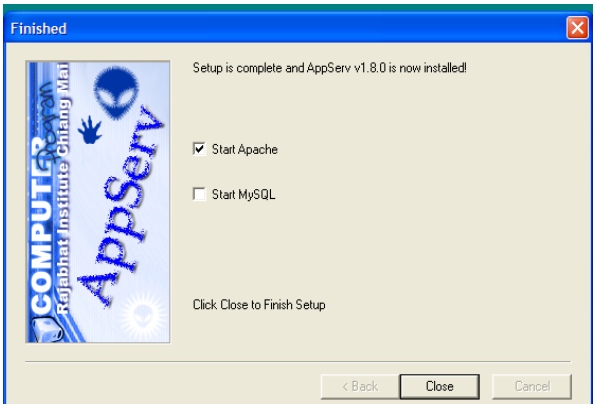


Figura 40. Se inicia el servicio de web Server.

Para cambiar parámetros de configuración del Apache, se lo debe hacer en el archivo de configuración httpd.conf, ya sea editando el archivo o ingresando a *inicio-> programas->appserv->Apache Configure Server ->Edit the Apache httpd.conf Configuration File.*

Para iniciar, detener, etc.. el servicio de Apache ejecutamos “stop, start, restart” usando *inicio-> programas->appserv->Apache Control Server ->(Stara, Stop, Restart).*

Instalación de Microsoft SQL Server 2000. Para la configuración se debe revisar la ruta de instalación del SQL, revisamos donde se encuentra el directorio “Data” (ubicación donde reside los data files de la base de datos) (Ej. E:\Archivos de programa\Microsoft SQL Server\MSSQL\Data) y en este directorio copiamos los archivos “hotel_Data.MDF” y “hotel_Log.LDF”, para tener toda nuestra estructura de la base de datos HOTEL, con una carga inicial de registros. Es importante que el sistema operativo tenga el sp3 como mínimo para poder instalar el SQL Server.

A continuación detallamos un ejemplo de la instalación y configuración de SQL 2000 Server:

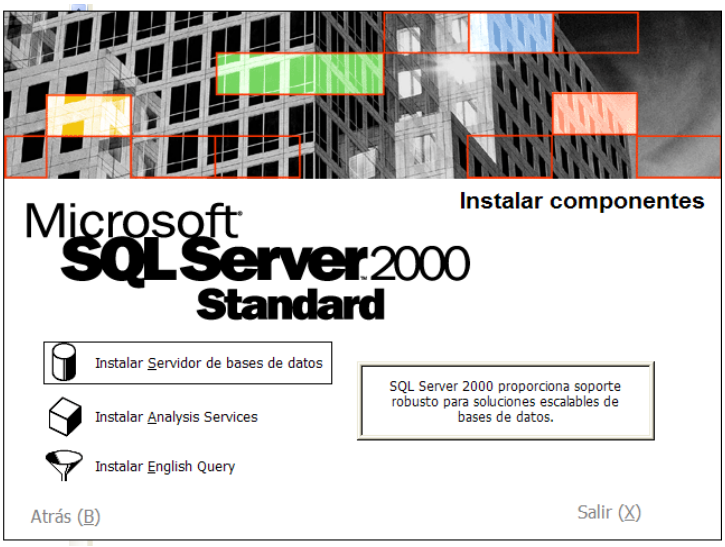


Figura 41. Pantalla de Instalación de SQL 2000 Server.

Se debe instalar una instancia nueva en el equipo local. Luego ingresamos un usuario y una contraseña para esta instancia. Instalamos la herramienta cliente y servidor, y el nombre de la instancia por defecto, seleccionamos instalación típica, y la ubicación donde van a residir el programa como tal y los datos (de cada una de las bases). Dejamos seleccionado *usar la misma cuenta para cada servicio* en la pantalla cuenta de servicios, e indicamos una contraseña para el usuario de dominio. Escogemos el modo de autenticación que sea administrado por el sistema operativo, y culmina la instalación de la instancia SQL como muestra el gráfico a continuación

Figura 42. Finalización de la instalación de SQL Server.

Terminada la instalación ejecutamos el *Administrador Corporativo* desde, Inicio->Programas->Microsoft SQL Server, y dejamos activo el servicio. Luego creamos una nueva base de datos con nombre "HOTEL", y copiamos los archivos "hotel_Data.MDF" y "hotel_Log.LDF", en el directorio de Datos de la base.

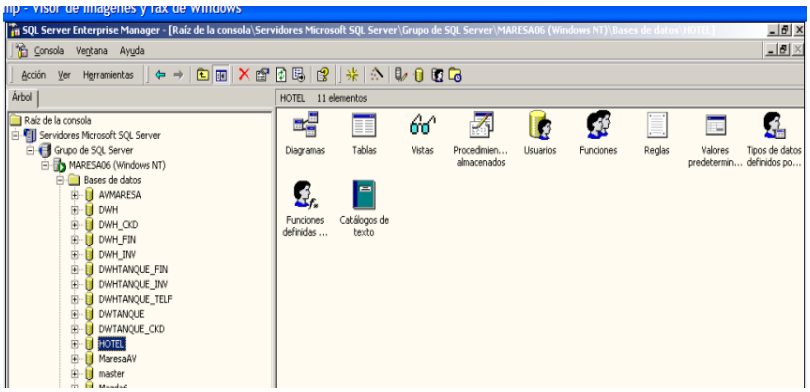


Figura 43. Creación de la Base de Datos "HOTEL"

CONCLUSIONES Y RECOMENDACIONES

1. Existen varias posibilidades a la hora de distribuir las reglas del negocio dentro de un esquema Cliente/Servidor. Sin embargo, sí hay ciertas pautas que se pueden tener en cuenta a la hora de tomar una decisión.

2. CORBA nos permite: Facilidad de implementación de las interfaces, facilidad de ampliación de funcionalidad, transparencia total respecto a la distribución (incluso excepciones), potencial enorme: multi-lenguaje, multi-arquitectura, multi-protocolo.

3. Interfaz dinámica frente a estáticas. La interface de las aplicaciones Web son netamente estáticas, pero existen formas de darles un funcionamiento dinámico. Con la distribución de componentes, se ha logrado que las interfaces sean livianas, de tal forma que el dinamismo dependa de la velocidad de comunicación, la capacidad de proceso de los servidores y el desempeño de los componentes.

   En base a esto podemos decir que las diferentes tecnologías de distribución apuntan a desarrollar herramientas que integren todas las capas de forma gráfica, brindando facilidades para el desarrollo de aplicaciones distribuidas con interfases altamente dinámicas.

4. La adopción de un diseño distribuido de aplicaciones empresariales, aumenta la re-usabilidad, reduce la cantidad de recursos, y los costos necesarios de desarrollo y mantenimiento.

5. Este nuevo enfoque de diseño pone en manos de los desarrolladores no sólo la funcionalidad que demandan las aplicaciones, sino también la seguridad, rapidez y flexibilidad.

ANEXOS

A continuación se muestra una lista de los anexos existentes, estos anexos ayudan al entendimiento completo del proyecto

Anexo 1: Manual del Usuario.

Describe el funcionamiento del sistema ventana por ventana. Permite visualizar el Layout completo de todas las ventanas, el documento es: ***ANEXO I Manual del Usuario.doc***

Anexo 2: Clases Cliente.

Muestra el código fuente del cliente, el documento es: ***ANEXO II CLASES CLIENTE.doc***

Anexo 3: Clases Servidor.

Muestra el código fuente del Servidor, el documento es: ***ANEXO III CLASES SERVIDOR.doc***

Anexo 4: Procedimientos del Servidor.

Aquí se muestra el código fuente de los procedimientos almacenados en la base de datos, el documento es: ***ANEXO IV PROCEDIMIENTOS ALMACENADOS.doc***

BIBLIOGRAFÍA

1. Robert Orfali, Dan Harkey, Client/ server Programimng with JAVA and CORBA (Second Edition, Jhon Wiley & Sons. Inc. Publishing, 1998)

2. Robert Orfali, Dan Harkey, Jeri Edwards,The essential Client/Server Survival Guide (Second Edition, Wiley Computer Publishuing, 1996)

3. R. Harkey, Edwards J., Instant Corba (First Edition; New York: John Wiley & Sons, Inc, 1997).

4. Jeri Edwards, Deborah DeVoe, 3-Capa Client/Server At Work, Wiley Computer Edition, 1997

5. Richard Monson-Haefel, Enterprise Java Beans (2nd Edition, O'Reilly, 2000).

6. Kenneth S. Rubin, Developing Object-Oriented Software An Experience-Based Approach, Prentice Hall PTR, 1997.

7. Santiago Comella-Dorda, John Robert, Robert Seacord, Kurt Wallnau, Enterprise Java Beans: A COSTS Architecture for Modern Enterprise Systems, (The 99 Software Enginnering Symposium, Carnegie Mellon University 1999), http://www.sei.cmu.edu/cbs/cbs_slides/99symposium/032tut.pdf, pp 1-80

8. Darleen Sadoski, Frank Rogers, 2 August 97, Client/Server Software Architectures-An Overview, http://www.sei.cmu.edu/str/descriptions/clientserver_body.html

9. Darleen Sadoski, Santiago Comella-Dorda , 16 Feb 2000, Three Capa Software Architectures, http://www.sei.cmu.edu/str/descriptions/threeCapa.html#34492

10. Darleen Sadoski, 2 January 97, Dos Capa Software Architectures http://www.sei.cmu.edu/str/descriptions/DosCapa.html