
# ESCUELA SUPERIOR POLITECNICA DEL LITORAL

## Facultad de Ingeniería en Electricidad y Computación

# "Tecnología Cliente Servidor con Arquitectura Corba"

### Sistema de una Empresa Telefónica

# Proyecto de Graduación

**Previo a la Obtención del Título de:**

INGENIERO EN COMPUTACION

**Presentado por:**

*Rodrigo Berrezueta Astudillo*

*Carlos Guzmán Becerra*

---

GUAYAQUIL - ECUADOR

1 9 9 9

# ESCUELA SUPERIOR POLITECNICA DEL LITORAL

**Facultad de Ingeniería en Electricidad y Computación**

# "Tecnología Cliente Servidor con Arquitectura Corba"

**Sistema de una Empresa Telefónica**

## Proyecto de Graduación

**Previo a la obtención del Titulo de:**

## Ingeniero en Computación

Presentado por:

**Rodrigo Berrezueta Astudillo.**

**Carlos Guzmán Becerra.**

Guayaquil – Ecuador

1999

# AGRADECIMIENTO

Al Ing. Carlos Valero:

Director de Tesis, por su ayuda

y colaboración para la realización

de este trabajo

# DEDICATORIA

A mis padres, hermanos, esposa, e hijo.

Rodrigo Berrezueta A.

A Dios, mis padres y mi hermano.

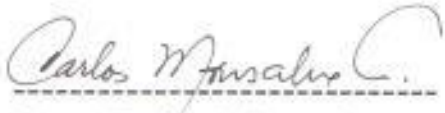
Carlos Guzmán B.

## DECLARACION EXPRESA

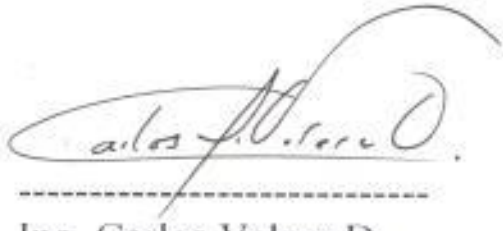
"La responsabilidad por los hechos, ideas y doctrinas expuestos en esta tesis, nos Corresponden exclusivamente; y el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL".

Rodrigo Berrezueta A.

Carlos Guzmán B.



Ing. Carlos Monsalve
Presidente del Tribunal



Ing. Carlos Valero D.
Director de Tópico

Ing. Guido Caicedo D.
Miembro de Tribunal

Ing. Rebeca Estrada
Miembro de Tribunal

# RESUMEN

Nuestro proyecto que aplica la tecnologia Cliente/Servidor, consiste en un sistema de "facturación de una empresa telefónica" que hace uso de dos nuevas tecnologias denominadas Corba y Java.

Los procesos intermedios entre el cliente y Servidor, que son necesarios para soportar sus interacciones, se denomina middleware. Dado que existen variadas formas de middleware y que muchos intentan unificar la forma de comunicación, es por lo que se establece un estándar de comunicación orientado a objetos. Este middleware Corba (Common Object Request Broker Architecture), es independiente de lenguajes de programación, asi como de sistemas operativos, debido a que utiliza una interface de lenguaje definida (IDL).

A pesar de ser independiente del lenguaje, Java es el lenguaje que mas se acopla a Corba, debido a que Java es conocido y distribuido en el ámbito mundial, además que es enfocado para realizar aplicaciones Web.

Nuestra aplicación trabaja con tecnología de Internet, pero fue realizado principalmente para que se ejecute en una Intranet debido a las opciones que ofrece.

# INDICE GENERAL

# INDICE DE FIGURAS

Pag.

# INTRODUCCION

El Conjunto de Procesos intermedios entre el Cliente y el Servidor, que cubre las necesidades de software para soportar las interacciones entre clientes y servidores, se lo denomina Middleware. Existen varios middleware en el mercado como son Dcom/ActiveX, RMI, Servlets, Corba, Sockets y HTTP/CGI.

Se intenta establecer un estándar en la forma de comunicación cliente/servidor por medio de objetos, para esto se realizaron especificaciones CORBA [Common Object Request Broker Architecture], dadas por la OMG (Object Management Group).

Corba brinda independencia de lenguajes y sistemas operativos, todo esto por medio de su IDL [Interface Definition Languaje] que permite la escritura de codigo, además de accesar a Objetos sin importar el Sistema Operativo o el lenguaje de programación en que se usen.

Nuestro *objetivo* es realizar la simulación de un sistema de facturación para una empresa telefónica en el que se muestre claramente la aplicación del lenguaje Java en conjunto con la arquitectura Corba.

Nuestra *hipótesis* es que la Arquitectura Corba es la mejor alternativa para la contracción de aplicaciones distribuidas en el Web. Segundo, el análisis y desarrollo

del proyecto, realizado con el método orientado a objetos, nos ayuda a modelar e implementar los procesos que negociaran entre cliente y servidor.

La *metodología* es realizar un análisis en cuestión, coleccionar información sobre como trabajan y facturan las empresas telefónicas, realizar un análisis orientado a objetos utilizando una combinación de metodológicas, hacer análisis del modelo cliente servidor, realizar la implementacion de objetos Corba, desarrollar la interface gráfica AWT de Java, unir interface visual con invocaciones a los objetos.

El *esquema de desarrollo*, consiste en la realización de un análisis y diseño orientado a objetos desde el punto de vista clásico, así todos los objetos que tengamos en nuestro análisis y diseño existirán en tiempo de compilación, tal que los clientes se comunicaran con estos objetos por medio de una solo interface Corba. Osea solo existirá una interface Corba para todos los objetos que residirán en el servidor y no será una interface para cada objeto.

Los procedimientos son de dos tipos internos y externos:

Externos se refiere a escoger un sistema operativo y lenguaje de programacion , así como los requerimientos y los recursos con los que se trabaja JDK1.1.5, Visibroker3.1, Visual Café 2.1, Netscape Communicator 6.0 y SQL 7.0.

Internos son Desarrollo del Análisis y diseño de la aplicación, Identificar y describir los modelos a construir, Definir que hará cada modelo, Desarrollo de los objetos.

*Capitulo I*

## TECNOLOGÍA CLIENTE SERVIDOR

### 1.2.1 DEFINICIÓN.

La tecnología Cliente/Servidor es la herramienta que nos ayuda en el análisis, diseño y creación de sistemas de software distribuido, en el cual los procesos servidores, son generalmente especializados (reduciendo la complejidad de los componentes individuales).

La complejidad de estas dos entidades Cliente y Servidor (lógicamente separadas) aumenta y actualmente incluyen: Procesos, Transacciones, Diseño de Base de Datos, Interfaz gráfica amigable, Objetos Distribuidos e Internet.

Físicamente son inter conectados por medio de una red local o de área extendida, utilizando uno o varios protocolos de comunicación.

### 1.2.2 EL MIDDLEWARE.

Es el Conjunto de Procesos Intermedios entre el cliente y el servidor, que cubre todas las necesidades de software para soportar las interacciones entre clientes y servidores.

El middleware no incluye el software como es la implementacion del servidor, ni la interfaz del usuario ni la lógica de la aplicación del cliente.

### 1.2.3 OBJETOS DISTRIBUIDOS

El hecho de que sus objetos pueden estar distribuidos en cualquier parte de la red significa que usted puede tener n-tier. En este concepto se tiene que cada objeto encapsula funcionalidad aplicable a una entidad en particular.

### 1.2.4 THREE TIER

Se puede definir una aplicación distribuida de tres capas (Three Tier) como aquella que consta de: primero la interfaz gráfica del cliente con su lógica de programación, segundo la lógica del funcionamiento del sistema y tercero los recursos que administra nuestro sistema de persistencia de datos como es la bases de datos SQL.

*Capitulo II*

# CORBA

## 2.1 Cliente /Servidor al estilo Corba.

La Arquitectura Común de requerimiento de objetos es conocido como (CORBA) Common Object Request Broker Architecture. Es él más importante y ambicioso proyecto middleware, producto de un consorcio llamado Object Management Group (OMG), que representa a mas de 800 compañias y donde la notable excepción es Microsoft quien tiene su propia tecnología middleware denominada Dcom.

Lo que hace a Corba tan importante es que define su middleware con el potencial de soportar independencia de cualquier tecnología existente. En otras palabras corba utiliza objetos con el fin de unificar la metáfora de traer aplicaciones existentes al bus Corba. Y lo mas importante es que las especificaciones del servicio están siempre separadas de la implementación.

Corba esta diseñado para tener componentes inteligentes que puedan descubrir e inter operar con cada uno de los otros componentes en el bus ORB.

Corba le permite crear objetos ordinarios, y luego hacerlos transaccionales, seguros y persistentes en tiempo de corrida. Permitiendo a los objetos pasar los parámetros, invocar su método y retornar los resultados.

## 2.2 Objeto Corba distribuido

Los objetos Corba son burbujas de inteligencia que pueden vivir en cualquier parte de la red. Son empaquetados como componentes binarios que su cliente remoto puede accesar por medio de invocación de métodos. Ambos el lenguaje y el compilador usados para crear objetos servidores son totalmente transparentes para el cliente.

## 2.3 IDL

EL secreto para el éxito de la OMG es crear especificaciones de Interfaces y no código.

Las especificaciones que utiliza Corba, son escritas en la Interface de Lenguaje neutral, mejor conocido como IDL (Interface Definition Languaje). Los componentes escritos en IDL son portables a través de Lenguajes,

herramientas, sistemas Operativos, y redes, debido a que el IDL provee independencia del sistema operativo y lenguaje de programación escogidos. El IDL de Corba es puramente declarativo, esto significa que no provee detalles de implementación.

El IDL puede ser escrito e invocado por cualquier lenguaje que provea enlazamiento Corba, así como C, C++, Ada, Smalltalk, COBOL y Java.

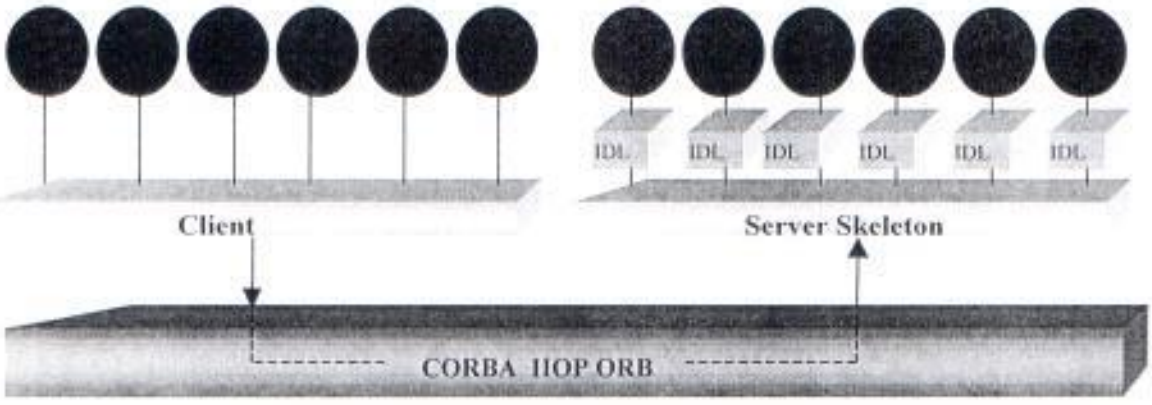



Figura 1 : Corba (IDL) sobre IIOP

## 2.4 ORB (Object Request Broker)

Es el bus de comunicación que permite a los objetos hacer requerimientos desde y hacia otros objetos localizados en forma local o remota, estableciendo la comunicación entre Clientes y Servidores por medio de Objetos.

El ORB es el middleware dentro de la arquitectura Corba, que nos permite comunicar a los Clientes con el Servidor y viceversa. El Cliente no necesita saber dónde reside el objeto, así como, no necesita saber como o en que lenguaje esta implementado el objeto servidor. Solo necesita conocer la interface del objeto.

## 2.5 Protocolo (IIOP)

La implementación de ORB trabaja sobre un protocolo que permite inter operatibilidad denominado Internet Inter ORB Protocol (IIOP), que es básicamente TCP/IP con mensajes de intercambio definidos por Corba, sirve como Protocolo Principal (Backbone). De esta manera cualquier propietario ORB puede conectarse con el Universo ORB haciendo requerimientos en el IIOP backbone.

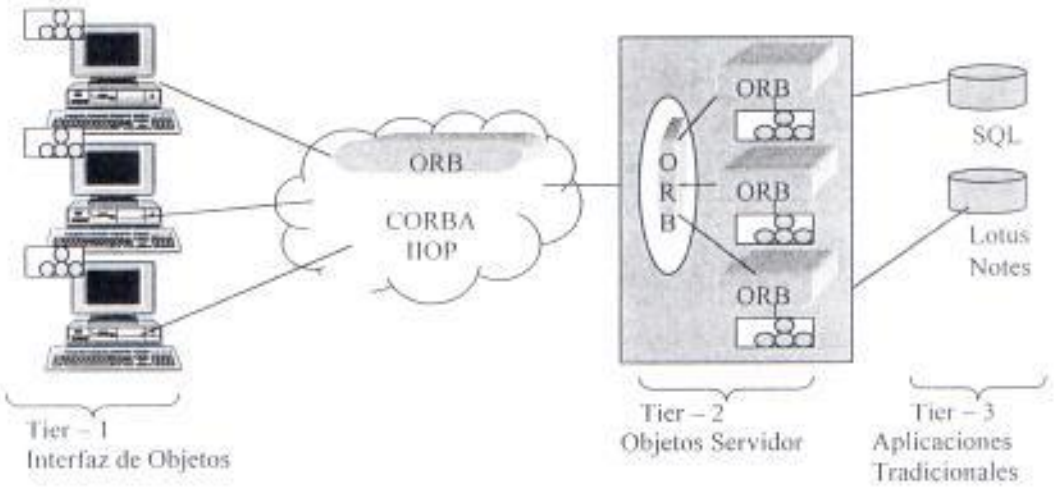



Figura 2: Funcionamiento de Corba Three Tier

ORB es mas sofisticado que otros middleware Cliente/Servidor incluyendo los tradicionales Remote Procedure Call (RPC), Message Oriented Middleware (MOM), Database Stored Procedure y Servicios Peer to Peer .

## 2.6 Applet/Java

El Applet es una pequeña aplicación pretendidamente corta basada en un formato gráfico sin representación independiente, que puede ser accesible desde un servidor Internet, que se transporta por la red, se instala automáticamente y se ejecuta como parte de un documento Web.

Los applet invocados por el cliente, viajan en la red en código de maquina virtual, al llegar a este, el browser interpreta este código y lo transforma a código binario para que el procesador pueda interpretar y ejecutar. Este paso que realiza el browser dependerá del sistema operativo así como del procesador. Por ejemplo en Win95/98/NT el browser convertirá el código para que lo entienda el procesador Intel.

La creación de los Applets de Java fue el primer paso en la creación de un Objeto Web en el sistema Cliente/Servidor. Pero Java necesitaba ser

complementado de una infraestructura de Objetos distribuidos, tal como es Corba.

## 2.7 Corba/Java

Corba se relaciona y enfatiza con transparencia en la red, debido a que provee una infraestructura de objetos distribuidos que le permite a las aplicaciones extender su alcance a través de redes, lenguajes, límites de frontera y sistemas operativos.

Java se relaciona y enfatiza con la transparencia en la implementación de la Interface debido a que provee una infraestructura de objetos portables que trabaja en todos los Sistemas Operativos Principales.

Por lo que se puede decir, Java inicia donde Corba termina. Ambos, Corba y Java se complementan el uno al otro.

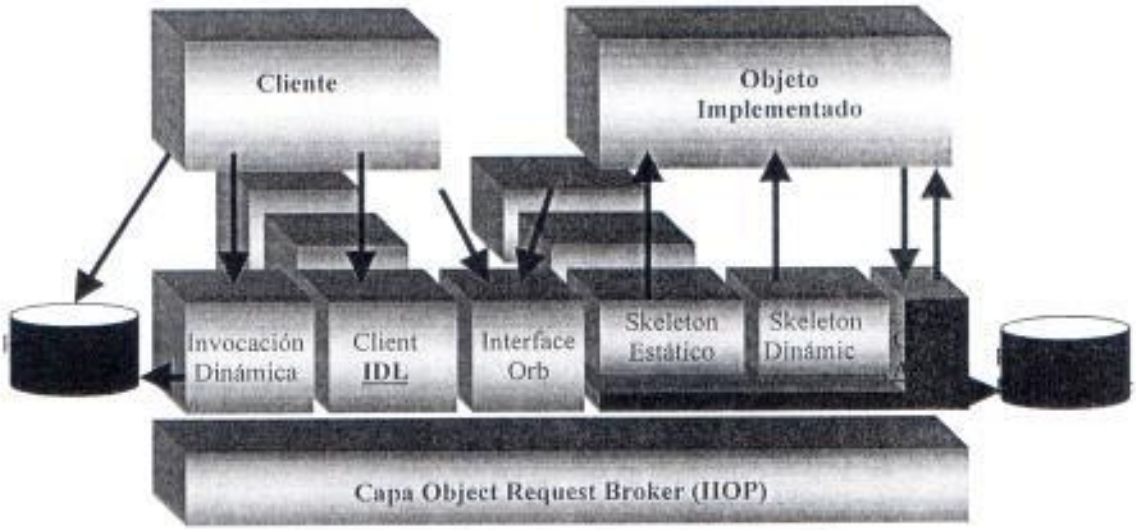



Figura 3: Invocaciones con Corba

## 2.8 Invocaciones estáticas

Una invocación Estática es igual a cualquier invocación a un método, pero por tratarse de una invocación remota con corba, este debe cumplir con lo siguiente: Se define el IDL que será nuestra estructura para la creación de la interface con que podrá el cliente comunicarse con el objeto que se encuentra en el servidor.

Después de definir el IDL, sé lo pre compila produciendo 6 archivos principales, de los cuales 3 son para el cliente Stub, Helper, Holder, y 3 son para el servidor Skeleton , Interface, Ejemplo para implementación de interface. A cada uno de estos archivo .java se los compila produciendo los archivos .class.

Luego se hace referencia a las definiciones de la clase en el repositorio de Interfaces, se registran los objetos en el repositorio de implementación y se crean instancias de los objetos en el servidor.

## 2.9 Portabilidad.

Para hacerlo portable hacemos uso del Naming Service. Este servicio de Nombramiento es como las páginas amarillas, donde se le permite encontrar objetos por medio de nombres. Cada objeto tiene una referencia única. La referencia se refiere a donde se encuentra ubicado este objeto.

*Capitulo III*

# APLICACIÓN DE TECNOLOGÍA CLIENTE SERVIDOR

## 3.1 Descripción del Proyecto

Se desea crear una aplicación para que sea accesible desde el Web, para lo cual se utilizará applets de Java. Los applets de Java podrán ser ejecutados desde distintas máquinas clientes.

El tema principal del proyecto es el desarrollo de un sistema cliente/servidor para una empresa telefónica usando tecnología de internet. Las máquinas clientes deberán contar con la existencia de browsers los cuales descargarán del servidor un applet de java en el cual se brindan las opciones de los distintos tipos de transacciones con los que contará el proyecto.

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

## 3.2 Funciones y Objetivos

El sistema de Facturación de la Empresa Telefónica tiene como objetivo:

- Permitir a los usuarios el manejo de las distintas opciones ofrecidas por el sistema.
- Mostrar un nuevo esquema de desarrollo de aplicaciones usando Corba y Java como tecnologías principales en la realización de proyectos que podrán ser ejecutados en una intranet o extranet y con la posibilidad de ciertas opciones ser ejecutadas en internet.

Para poder cumplir con estos objetivos se dispone de las siguientes opciones:

- **Ingreso**. Esta transacción es para poder ingresar el consumo de un cliente a la Base de datos.
- **Consulta**. Es para mostrar al cliente cual es el consumo que ha tenido, mostrando totales de consumo, así como deudas pendientes y detalle de llamadas .
- **Recepción de Pagos**. Es para realizar el pago de lo adeudado por el servicio telefónico prestado. Este se lo puede hacer en efectivo o en cheque.

- **Detalle de Consumo**. Esta transacción nos muestra como su nombre lo indica, el detalle del Consumo, siendo local, nacional, regional, celular, internacional,
- **Asignación de Números**. Es para la asignación de números a nuevos abonados o clientes.
- **Baja de Números.** Es una transacción para retirar de su uso a usuarios, debido a problemas ya sea monetario o personal del cliente.

## 3.11 Arquitectura general de la Aplicación

Es una arquitectura basada en la Arquitectura Cliente/Servidor.

El usuario accesa por medio de una pagina HTML al Applet que trae consigo partes de Corba que permiten crear el puente ORB y así operar con objetos (Corba) en la red.

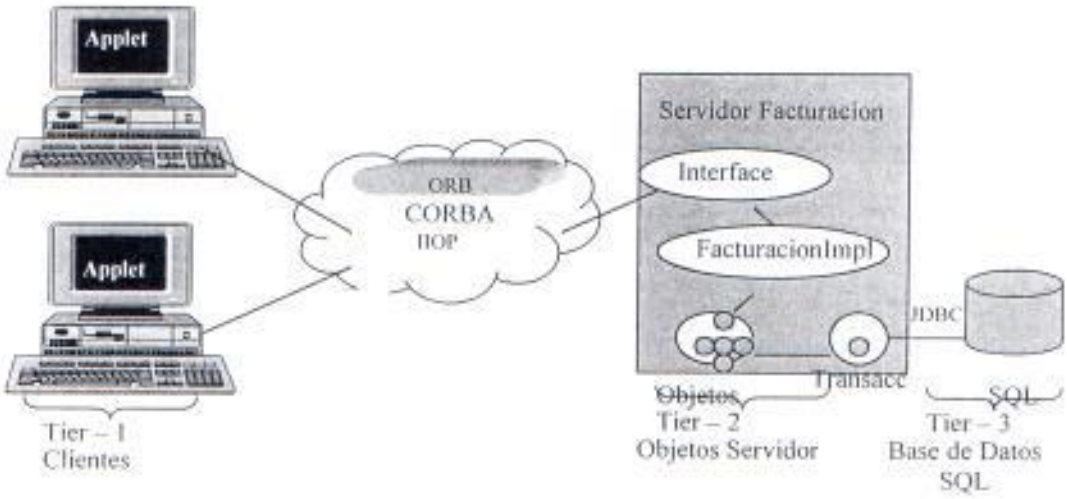



Figura 4: Aplicación vista en Three Tier

# Capitulo IV

# HERRAMIENTAS UTILIZADAS

## 4.1 Visual Café

Es una herramienta que sirve para crear los applet en forma Visual, muy parecido a Visual Basic. Nosotros hemos utilizado un applet en el que se introdujo el panel principal y 1 tab panel que contiene 6 paneles.

La versión es Symantec Visual Café 2.1 for Java trial versión. Indicando que nos dura Visual Café por un mes. Esta es la segunda generación de herramientas de programación que incluye Java Beans.

Las invocaciones a objetos Corba se la realiza en forma manual, ósea no existe programación visual para ello.

En cambio en Visual Age se dispone de una manera de invocar objetos Corba visualmente, pero que no es perfecta según los autores del libro que estamos usando de referencia, aparte de esto el programa demanda el uso de muchos

recursos y define un esquema de programación diferente por lo que preferimos usar Visual Café.

## 4.2 Visibroker

Visibroker incluye un compilador IDL que genera código java, tanto el skeleton para el servidor como el stub para el cliente.

Visibroker para Java es el fabricante del orb que soporta el estándar CORBA 2.0 y es la herramienta que nos ofrecerá una serie de utilerías y herramientas para que nuestro programa trabaje con Corba.

Los objetos construidos con Visibroker son fácilmente accesibles por las aplicaciones Web, ya que se comunica usando el Internet Inter-ORB Protocol (IIOP), que es el estándar para la comunicación entre y a través de objetos distribuidos.

Visibroker soporta métodos de invocaciones corba tanto estáticos como dinámicos. También incluye un repositorio de interfaces Corba, escritos en Java. Todo esto debido a que se ajusta perfectamente al estándar Corba.

## 4.3 OS Agent

Este proceso inicia el Smart Agent, que provee de un servicio de búsqueda de objetos y detección de fallas al iniciar una aplicacion.

El ORB Smart Agent (osagent) es un servicio de directorio dinámico y distribuido que provee facilidades para las aplicaciones de los clientes y objetos implementados. Cuando una aplicación de cliente invoca el método bind (enlazar) sobre un objeto, el osagent localiza el objeto y la implementación especifica, y así establece la conexión entre el cliente y objeto servidor.

Los objetos Implementados se registran con el osagent para que las aplicaciones del cliente puedan localizar y usar esos objetos.

## 4.4 Gatekeeper

Este comando inicia el IIOP gatekeeper.

El gatekeeper habilita a los Applets, para comunicarse con objetos servidores a través de largas redes mientras mantienen conformidad con restricciones de seguridad impuestas por el Web Browser. El Gatekeeper puede actuar como servidor Web para HTML y otros tipos de archivos.

Permite al Applet invocar al objeto servidor en un host distinto del que se encuentra el Applet originalmente. Permite al servidor invocar métodos en el Applet vía callbacks.

Provee HTTP tunneling para permitir a los clientes comunicarse con objetos servidores a través de firewalls que no soportan IIOP.

## 4.5 Naming Service

Servicio Visibroker que permite a las aplicaciones cliente atar objetos usando nombres significativos y lógicos sin tener que direccionar por medio de cualquier convención de nombres de una plataforma especifica.

Con el Naming Service, el código es totalmente portable a través del los diferentes ORB's de Java Comerciales.

Para hacerlo portable se debe:

1)Remplazar el Visigenic specific *bind por* el servicio de nombramiento Corba.

2) Reemplazar Basic Object Adapter (BOA) con la semántica del Portable Object Adapter (POA).

El Servicio de Nombramiento mapea o guarda la relación entre nombres y las referencias a un objeto. La asociación del nombre al objeto es llamada name *binding*.

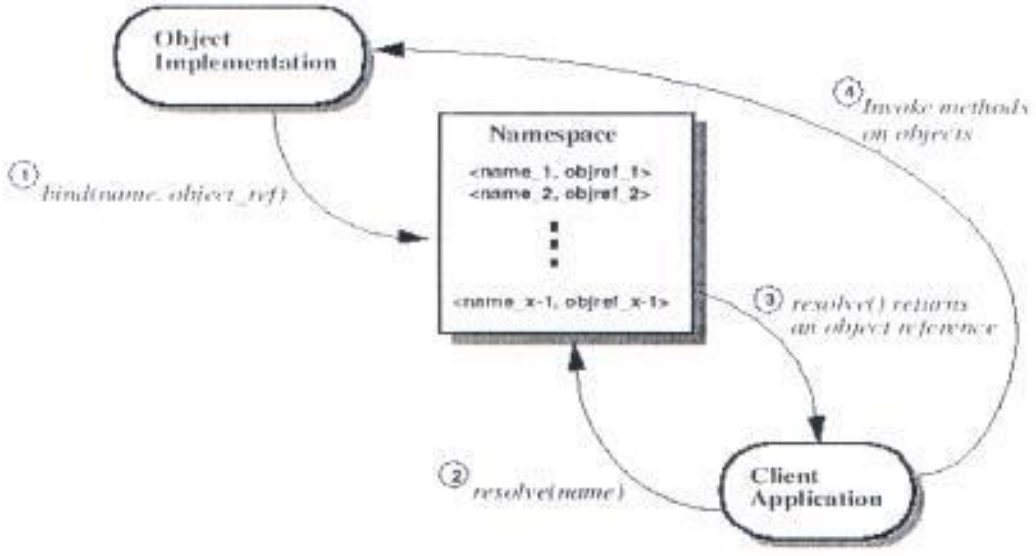



Figura 5: Enlazando, resolver y usar nombre de objetos para un nombre de contexto

El Servicio de nombramiento utiliza el namespace de la siguiente manera:

1.- La implementacion de un Objeto enlaza un Nombre a un Objeto dentro del namespace. bind(name, object_ref)

2.- La aplicación cliente usa el mismo namespace para resolver el nombre.

Resolve(name)

3.- Obtener una referencia del objeto.

Resolve( ) Retorna al cliente el resultado de la referencia del objeto

4.- Aplicación del cliente conociendo la referencia del objeto puede invocar métodos en el Objeto.

## 4.6 Java Developer Kit

Existen varias versiones en el mercado, de las cuales hemos escogido la versión 1.1.5 que es el que acompaña al CD del libro Java/Corba.

Java Developer Kit es conocido mejor como JDK que representa la implementación base de Java Soft, que permite que Java trabaje mas eficientemente y seguro en ambientes distribuidos.

Las grandes mejoras son las incorporaciones hechas al AWT que permite mas facilidades, pero lo mejor de todo es que Java en versión 2.0 ya viene con Corba ósea con el Compilador IDL.

# Capitulo V

## SIMULACIÓN DEL PROCESO DE FACTURACIÓN

### 5.1 Análisis y Diseño Orientado a Objetos

La simulación del proceso de facturación, se la realiza haciendo una descripción lo más precisa posible de lo que debería realizar el sistema, esto es para el análisis y diseño Orientado a Objetos; esta descripción nos permite crear un modelo que nos defina los detalles de la implementación. Se utilizan metodologías para esta descripción. Los modelos utilizados fueron: Casos de Uso, Escenarios, y Diagrama de Interacción de Objetos.

La idea es crear una aplicación Cliente/Servidor 3 Tier para usarlo en el Web usando Corba y Java.

## 5.2 Modelo de Requerimientos.

Se refiere a la forma de comportamiento externo que debe tomar el sistema cuando este terminado y puesto en funcionamiento.

En nuestro caso el sistema de facturación de una empresa telefónica debe permitir a los empleados hacer varias transacciones fácilmente en un ambiente amigable para que sea ejecutado en una intranet o extranet, así como con pequeñas variaciones hacer que este servicio sirva para que los clientes puedan hacer consultas por medio de la red internet.

### 5.2.1 Requerimientos Funcionales

Son los requerimientos observables de lo que hará el sistema, para lo cual se utiliza los casos de uso.

Primero partiremos del análisis de los requerimientos del sistema, ya que el análisis del sistema se refiere a la abstracción precisa de que debe hacer el sistema.

### 5.2.1.1 Análisis del sistema.

El proyecto ha sido concebido para ejecutarse en una intranet o internet, por lo que el programa deberá utilizar applets y ejecutarse en un browser. Además este applet debe proveer la interfaz gráfica necesaria para brindarle al usuario la posibilidad de realizar las siguientes 5 transacciones:

De lo revisado anteriormente y del análisis de la información recogida, se identifican las siguientes metas "primordiales" del sistema.

Ingreso de Consumo

Consulta

Recepción de pagos

Asignación de Números

Baja de números

### 5.2.1.2 Actores:

Los actores son definidos como entidades que interactúan con el sistema. Pueden ser de dos tipos:

#### 5.2.1.2.1 Primarios:

Los actores primarios son definidos como aquellos que activamente están interactuando con el sistema, el sistema deberá satisfacer sus requerimientos.

#### 5.2.1.1.2 Secundarios:

Son las personas intermediarias ó sistemas externos, que logran las metas del actor primario, soportan los objetivos del actor primario.

### 5.2.1.3 Casos de Uso:

Los casos de uso son una metodología que describe el funcionamiento del sistema en un alto nivel. Nuestros casos de uso irán descritos en un formato de tabla y representarán las diferentes situaciones del sistema.

#### 5.2.1.3.0 Definición de Casos de Uso

A continuación se describen los diferentes casos de uso adoptados para la descripción del funcionamiento de nuestro sistema.

#### 5.2.1.3.1 Verificar Numero en Principal

| | | |
|---|---|---|
| Meta | | Verificar si el Numero esta asignado a Cliente. |
| Alcance y Nivel | | Permitir al Empleado accesar a datos del Cliente |
| PreCondiciones | | Numero teléfono exista y este asignado. |
| PostCondiciones de éxito | | Muestra datos del Cliente |
| PostCondiciones de fallo | | Muestra mensaje "No existe Numero en base de datos" |
| Actores Primarios | | Empleado |
| Actores Secundarios | | Objeto Cliente que verifica asignación de numero telefónico. |
| Trigger | | El botón del Applet llama a getdatos Cliente enviando numero telefónico. |
| Descripción | Paso | Acción |
| | 1 | Ingresar número en panel Principal. |
| | 2 | Presionar Aceptar para que este busque datos del Cliente. |
| Extensiones | Paso | Acción |
| | 1ª | Se debe ingresar Dígitos no letras y él numero debe ser de 6 dígitos. |
| | 2ª | Se puede hacer Clic con el ratón o presionar Enter con el teclado. |
| Variaciones | Paso | Acción |
| | 2ª | Si él número ingresado no existe en la base de datos entonces mostrar mensaje de error. |

### 5.2.1.3.2 Ingreso de Consumo

| | |
|---|---|
| Meta | Ingresar el consumo de un mes |
| Alcance y Nivel | Permitir al Empleado hacer ingresos |
| PreCondiciones | Numero teléfono exista y este asignado. |
| PostCondiciones de éxito | Ingreso de consumo en la base de datos exitoso |
| PostCondiciones de fallo | Existe falla en transmitir a la base de datos |
| Actores Primarios | Empleado |
| Actores Secundarios | Llamar a función para que envíe detalles o datos del Consumo. |
| Trigger | El Boto de Aceptar en Ingreso llama a función Enviar Detalle. |

| Descripción | Paso | Acción |
|---|---|---|
| | 1 | Ingresar a panel de Ingreso. |
| | 2 | Escoger tipo llamada en el Choice, (Local, Regional, Nacional, Celular) o Internacional. |
| | 3 | Ingresar datos de llamada. Si es tipo Local: Telf Destino, Hora de llamada, Minutos, Fecha de llamada. |
| | 4 | Si es Internacional se ingresa lo anterior mas el Destino Intencional, y Destino. |
| | 5 | Si es Internacional, se escoge según región Internacional (Destino). |
| | 6 | Presionar Aceptar. Se debe mostrar mensaje de ingreso exitoso. |
| Extensiones | Paso | Acción |
| | 2a | Se ingresa el consumo telefónico y se verifica su existo guardando en la base de datos. |
| | 4a | Cuando se escoge Internacional se habilita el Choice de Internacional y Textfield de Destino. |
| Variaciones | Paso | Acción |
| | 2a | Si no se ingresan todos los datos debería mostrar mensaje de Error. |

#### 5.2.1.3.2 Consulta y Detalle de consumo

| | |
|---|---|
| Meta | Poder ver el Consumo Telefónico por mes. |
| Alcance y Nivel | Permitir ver el consumo telefónico, (total a pagar, pendiente, etc.) |
| PreCondiciones | Numero teléfono existe y está asignado. |
| PostCondiciones de éxito | Muestra en pantalla el consumo. |
| PostCondiciones de fallo | Numero de teléfono no existe |
| Actores Primarios | Empleado |
| Actores Secundarios | Función que lleve datalle de Consumo |
| Trigger | El botón Aceptar de Consulta llama a la función encargada de obtener estos datos. |

| Descripción | Paso | Acción |
|---|---|---|
| | 1 | Ingresar al Panel Consulta. |
| | 2 | Se muestra los datos de Consulta para el mes de Facturación , si desea ver otro mes debe Ingresar nueva fecha a Consultar. |
| | 3 | Presionar botón Aceptar. Luego se muestra en pantalla el Consumo Total del mes que desea. |
| | 4 | Si desea ver otra fecha debe ingresar nueva fecha de consulta y regresar al paso 3. |
| Extensiones | Paso | Acción |
| | 2a | Si los datos no existen, los cuadros donde se muestran los datos saldrán en blanco. Si se intenta ver mes superior al de facturación no saldrán datos. |

### 5.2.1.3.4 Recepción de Pagos

| | |
|---|---|
| Meta | Permitir receptar pagos de los clientes. |
| Alcance y Nivel | Mostrar el consumo total y permitir cancelar este consumo. |
| PreCondiciones | Numero teléfono exista y esté asignado. |
| PostCondiciones de éxito | Cliente logra pagar el consumo de mes. |
| PostCondiciones de fallo | No logra hacer las transacciones de pago. |
| Actores Primarios | Empleado |
| Actores Secundarios | Verificación asignado, verificación de número y Pago mayor a cero |
| Trigger | Presionar botón de Recepción de Pagos |

| | | |
|---|---|---|
| Descripción | Paso | Acción |
| | 1 | Ingresar a Panel de Recepción de pagos. |
| | 2 | Se muestra el consumo total del mes. Así como el valor pendiente. |
| | 3 | Se ingresa la cantidad que se desea cancelar. Debe ser mayor a cero. |
| | 4 | Se escoge del Checkbox el tipo de pago Efectivo o Cheque. Si es Cheque se muestra cajas par ingresar numero de Cheque y Banco. |
| | 5 | Se presiona Aceptar. Se muestra mensaje de pago exitoso. |
| Extensiones | Paso | Acción |
| | 2ª | Si no se ingresa datos en Total a pagar no se hará acceso a la base de datos solo se hará en el Applet mostrando el mismo valor pendiente. |
| Variaciones | Paso | Acción |
| | 2ª | Solo se pueden hacer pagos del mes que se facturó. |

### 5.2.1.3.5 Asignación de Números

| | |
|---|---|
| Meta | Asignar números disponibles a nuevos clientes. |
| Alcance y Nivel | Crear relación haciendo la asignación de un numero a un Cliente. |
| PreCondiciones | Que exista disponible numero telefónico. |
| PostCondiciones de éxito | Muestre en pantalla un mensaje exitoso de haber asignado numero. |
| PostCondiciones de fallo | No existen numero disponibles o Datos del Cliente no se han ingresado correctamente. |
| Actores Primarios | Empleado |
| Actores Secundarios | Función que obtenga los números disponibles y función que guarde los datos del Cliente con el numero. |
| Trigger | Al botón Aceptar de Asignación llamar a función guardar Cliente. |

| Descripción | Paso | Acción |
|---|---|---|
| | 1 | Ingresar en panel Asignación de Números. |
| | 2 | Escoger en Checkbox el numero de Area |
| | 3 | Al presionar Mostrar, este muestra los números disponibles según el numero de Area. |
| | 4 | Escoger en ventana el numero que desee. |
| | 5 | Llenar los datos del Cliente en panel Principal. |
| | 6 | Si están llenados los datos del Cliente presionar Aceptar. |
| Extensiones | Paso | Acción |
| | 2ª | Se habilita botón Mostrar cuando se escoge numero de Area. |
| | 3ª | Por omisión se escoge el primer numero. |
| | 5ª | Si no están todos los datos del Cliente este muestra mensaje indicando que llene correctamente datos del Cliente. |
| Variaciones | Paso | Acción |
| | 3ª | No se realiza ninguna asignación hasta que no se presione Asignar. |

### 5.2.1.3.6 Baja de Números

| | |
|---|---|
| Meta | Realizar Baja de Números |
| Alcance y Nivel | Quitar la asignación de un numero a un usuario. |
| PreCondiciones | Solicitado por escrito la baja del Cliente. |
| PostCondiciones de éxito | Numero queda sin asignación ósea queda disponible. |
| PostCondiciones de fallo | No se culmino con dar de baja a Cliente. |
| Actores Primarios | Empleado |
| Actores Secundarios | Se elimina numero telefónico de Cliente. |
| Trigger | Se llama a función eliminar. |

| Descripción | Paso | Acción |
|---|---|---|
| | 1 | El numero a dar de Baja es el ingresado en Principal. |
| | 2 | Si desea modificarlo presione cambiar. |
| | 3 | Si desea dar de Baja presione Eliminar. |
| | 4 | Se muestra un mensaje de Advertencia indicando que el numero será dado de Baja, Si esta seguro presione otra vez el Botón Eliminar. |
| Extensiones | Paso | Acción |
| | 2a | Si presiono por primera vez eliminar y luego presiona Cancelar, entonces el numero no será dado de Baja . |
| | 2b | Si se da de Baja se muestra el mensaje de que el Siguiente numero que perteneció al Cliente fue dado de Baja. |

### 5.2.1.4 ESCENARIOS:

Los escenarios nos describen las variaciones del comportamiento de los casos de uso. Un caso de uso específico puede desenvolverse en diferentes escenarios y cada uno de estos escenarios nos devolverán una respuesta determinada.

Los escenarios nos darán una idea más clara de cómo los objetos se comportarán en situaciones diferentes.

En nuestro análisis hemos resuelto una manera sencilla pero eficiente para declarar escenarios, esta consistirá en la declaración de dos tipos de escenarios para diferentes situaciones que se den en los casos de uso: operación éxitosa y operación fallida. Una situación especifica puede tener más de una operación éxitosa o más de una operación fallida.

#### 5.2.1.4.1 Lista de casos de uso para la elaboración de escenarios.

- Verificar Numero en Principal.
- Ingreso de Consumo.
- Consulta.

- Recepción de Pagos.
- Detalle de Consumo.
- Asignación de Números.
- Baja de Números.

**ESPECIFICACIONES DE ESCENARIO**

**5.2.1.4.3Caso de Uso Verificar Numero en Principal.**

**SUPUESTOS**

- Se ingresa el número de teléfono del Cliente para verificar el número y mostrar datos de este.
- El número esta correctamente ingresado. (Solo se ingresan 6 dígitos). y
- El número existe y está Asignado. (El número esta Asignado en la base de datos SQL).
- Se llama a alguna función que obtenga los datos del cliente y los devuelva.

**RESULTADOS**

Se retornan los datos del Cliente, para ser mostrados los datos en panel Principal.

Se realizan las transacciones de Consulta, Detalle de llamadas y Recepción de Pagos, para luego ser mostrados, estos datos en sus respectivas ventanas o paneles.

**SUPUESTOS**

- Se ingresa el número de teléfono del cliente para verificarlo y mostrar los datos de este.
- El número no está correctamente ingresado . (Se ingresan letras o incorrecto numero de dígitos).
- El número existe y está asignado. (El número está asignado en la base de datos SQL).

**RESULTADOS**

- Se presenta un mensaje de error indicando que se han ingresado letras.
- Las operaciones se las realiza dentro del Applet no se llama al servidor.

#### 3.2.1.3.3 Operación fallida no existe número.

**SUPUESTOS**

- Se ingresa el número de teléfono del cliente para verificar el número y mostrar datos de este.
- El número esta correctamente ingresado.
- El número no existe. (El número no está asignado en la base de datos SQL).

**RESULTADOS**

- Se presenta un mensaje error indicando que el número no existe.
- No se muestran datos y no se realizan las transacciones para Consulta, Recepción de Pagos, Detalle de Consumo.

**ESPECIFICACIONES DEL ESCENARIO**

**5.2.1.4.4 CASO DE USO INGRESO DE CONSUMO.**

5.2.1.4.4.1 Ingreso con éxito

**SUPUESTOS**

- Se ingresa todos los datos del Consumo del Cliente, Iniciando con el tipo de llamada, Local, Nacional, Regional, Celular o Internacional.
- Dependiendo del tipo de ingreso de tipo de llamada un tipo determinado de transacción.
- El cliente presiona Aceptar para llamar a la función Enviar Detalle enviando los datos de lo que se desea ingresar en la transacción Ingreso.

**RESULTADOS**

- Se guarda en la base de datos los detalles de: tipo de llamada, hora llamada, minutos, etc.
- Se muestra en pantalla un mensaje indicando el correcto procesamiento de datos. El Ingreso ha sido guardado con éxito.

5.2.1.4.2 Ingreso fallido faltan datos.

**SUPUESTOS**

- Falta Ingresar datos del Consumo del Cliente, Iniciando con el tipo de llamada, Local, Nacional, Regional, Celular o Internacional.
- El cliente presiona Aceptar

**RESULTADOS**

- Se muestra en pantalla un mensaje indicando que falta Ingresar datos.
- Las operaciones se las realiza dentro del Applet no se llama al servidor.

## ESPECIFICACIONES DEL ESCENARIO

### 5.2.1.4.5 CASO DE USO CONSULTA.

#### 5.2.1.4.5.1 Consulta exitosa.

**SUPUESTOS**

- Se muestran en pantalla los datos de Consumo en el Mes de Facturación. Como son: Total del mes, deuda, Impuestos, etc.
- Si desea consultar otro mes, se tiene que cambiar la fecha de Consulta, en el formato mes/año.
- El cliente presiona Aceptar para llamar a alguna función que devuelvan los datos requeridos.

**RESULTADOS**

- Se muestra en pantalla los datos de la consulta con el mes que se ingresó.

3.2.1.1.5.2 Consulta fallida formato incorrecto.

**SUPUESTOS**

- Se muestran en pantalla los datos de consumo en el mes de facturación, como son: Total del mes, deuda, Impuestos, etc.
- Si desea consultar otro mes, se cambia la fecha de Consulta en formato incorrecto.
- El cliente presiona aceptar.

**RESULTADOS**

- Se muestra en pantalla un mensaje de error indicando que ha sido mal ingresada la fecha de consulta.
- Las operaciones se las realiza dentro del Applet no se llama al servidor.

### 2.1.4.5.3 Consulta fallida fecha posterior.

**SUPUESTOS**

- Se muestran en pantalla los datos del consumo en el mes de facturación, como son: Total del mes, deuda, Impuestos, etc.
- Si desea consultar otro mes, se cambia la fecha con una fecha posterior a la fecha de facturación.
- El cliente presiona Aceptar.

**RESULTADOS**

- Se muestra en pantalla un mensaje de error indicando que la fecha es mayor a la fecha de facturación.

## ESPECIFICACIONES DEL ESCENARIO

## 5.2.1.4.6 CASO DE USO RECEPCION DE PAGOS.

5.2.1.4.6.1 Recepción exitosa.

**SUPUESTOS**

- Se muestran en pantalla el valor de consumo del mes de facturación y deuda pendiente.
- Se ingresa la cantidad que se pagará. Si no cubre la totalidad, el saldo queda como deuda para el siguiente mes.
- El cliente presiona Aceptar para enviar el valor de pago del Consumo, llamando a la función registrar pago, que a su vez llama al procedimiento de ingreso del pago, así como el saldo o deuda.

**RESULTADOS**

- Se guarda en la base de datos el ingreso de pago y deuda pendiente.
- Se muestra en pantalla los valores actualizados de los casilleros que contienen total a pagar, deuda y total.

4.1.1.6.2 Recepción fallida no ingresa dato.

**SUPUESTOS**

- Se muestran en pantalla el valor de Consumo del mes de facturación y Deuda pendiente..
- No se ingresa la cantidad que se pagara.
- El cliente presiona Aceptar para enviar pago de Consumo.

**RESULTADOS**

- Se muestra en pantalla los datos actualizados realizandose esto en el cliente sin llamar al servidor.

## ESPECIFICACIONES DEL ESCENARIO

## 5.2.1.4.7 CASO DE USO DETALLE DE LLAMADAS.

5.2.1.4.7.1 Detalle exitoso.

**SUPUESTOS**

- Se muestran en pantalla dentro de una lista los datos detallados de consumo en el mes de facturación.
- Si desea consultar otro mes, se tiene que cambiar la fecha de Consulta, en el formato mes/año.
- El cliente presiona Aceptar para llamar a alguna función que obtenga los datos de las llamadas.

**RESULTADOS**

- Se muestra en pantalla los detalles de las llamadas en la consulta con el mes que se ingresó.

### 3.1.3.2 Detalle Fallido.

**SUPUESTOS**

- Se muestran en pantalla dentro de una lista los datos detallados de consumo en el mes de facturación.
- Para consultar otro mes, se cambia la fecha de consulta, en formato equivocado.
- El cliente presiona Aceptar.

**RESULTADOS**

- Se muestra en pantalla un mensaje de error indicando que la fecha de consulta esta mal ingresada.
- Las operaciones se las realiza dentro del Applet no se llama al servidor.

## ESPECIFICACIONES DEL ESCENARIO

## 5.2.1.4.8 CASO DE USO ASIGNACION DE NUMEROS.

### 5.2.1.4.8.1 Asignacion Exitosa.

**SUPUESTOS**

- Se escoge un número de teléfono según un código de área específico. Para esto se escoge el área, luego se presiona mostrar números, y se escoge un número.
- Luego se ingresa todos los datos del cliente al que se desea asignar el número telefónico.
- El cliente presiona asignar para llamar a la función que guarde los datos del cliente .

**RESULTADOS**

- Se guarda en la base de datos los datos del Cliente y el número que se le asigna. Pasando este numero a no estar disponible.
- Se muestra en pantalla un mensaje indicando el correcto procesamiento de datos. La Asignación ha sido guardada con éxito.

4.1.1.2 Asignación fallida faltan datos.

**SUPUESTOS**

- Se escoge un número de teléfono según un código de área especifico. Para esto se escoge el área, luego se presiona mostrar números, y se escoge un número. Luego No se ingresan Todos los datos del Cliente al que se desea asignar el numero telefónico.
- El cliente presiona Asignar

**RESULTADOS**

- Se muestra en pantalla un mensaje de error indicando que faltan datos del Cliente.
- Queda habilitado para terminar de llenar los datos que faltan y así poder hacer de este una transacción éxitosa.
- Las operaciones se las realiza dentro del Applet no se llama al servidor.

### 3.1.1.1.3 Asignación Fallida no disponibles.

**SUPUESTOS**

- Se escoge un número de teléfono según un código de área específico. Para esto se escoge el área, luego se presiona mostrar números, y se escoge un número.
- No existen números disponibles.
- El cliente presiona Asignar

**RESULTADOS**

- No se muestran números.
- No se puede ingresar datos de cliente a asignar.

## ESPECIFICACIONES DEL ESCENARIO

## 5.2.1.4.9 CASO DE USO BAJA DE NUMEROS.

### 5.2.1.4.9.1 Baja Exitosa.

**SUPUESTOS**

- El número a dar de baja es el que se ingresó en el panel Principal,
- Se presiona dar de baja.
- Se muestra un mensaje "Está seguro que quiere dar de baja a este número".
- La respuesta es Si.

**RESULTADOS**

- Se da de Baja al cliente.
- Se muestra mensaje indicando el número y datos del Cliente dado de Baja.

### 3.3.1.3.2 Baja cambio de número.

**SUPUESTOS**

- El número a dar de baja es el que se ingresó en el panel Principal.
- Se presiona cambiar de número.
- Se escribe este en el casillero ubicado en Principal, y se presiona Aceptar verificando la existencia del nuevo número, si es exitoso entonces se muestran los datos.
- Así, está listo para ser dado de baja a otro número.

**RESULTADOS**

- Se cambia de número a dar de baja.
- Las operaciones se las realiza dentro del Applet no se llama al servidor.

5.7.1.9.3 Baja Fallida.

**SUPUESTOS**

- El número a dar de baja es el que se ingresó en el panel Principal,
- No se presiona Dar de Baja. O
- Se presiona Dar de baja. Se muestra un mensaje "Esta seguro que quiere dar de baja a este numero".
- La respuesta es No.

**RESULTADOS**

- Se cancela la operación de dar de baja.
- Las operaciones se las realiza dentro del Applet no se llama al servidor.

### 5.2.2 Requerimientos no Funcionales.

Se refiere a las limitaciones del sistema y/o criterios como la confiabilidad, el rendimiento, el costo, la utilidad que tendrá.

- Los Applets no funcionan en browser's antiguos. El sistema esta hecho para funcionar en una Intranet o internet con browser actualizado (4.*).
- El sistema de red deberá estar implementado con el protocolo TCP/IP.
- La implementación del sistema preferiblemente deberá realizarse en una máquina con las mejores prestaciones posibles.
- La aplicación tarda un poco en cargarse pero una vez realizado esto todas las funciones se ejecutarán rápidamente.

## 5.3 Modelo de Análisis y Diseño.

El modelamiento dinámico describe la evolución temporal de los objetos en un sistema.

El sistema es una entidad cuyo comportamiento puede ser descrito por los casos de uso.

### 5.3.1 Modelo Dinámico

## 5.3.1.1 Flujo de Ventanas y Layouts.

Layout de Principal


Figura: Layout de Principal

Layout de Ingreso

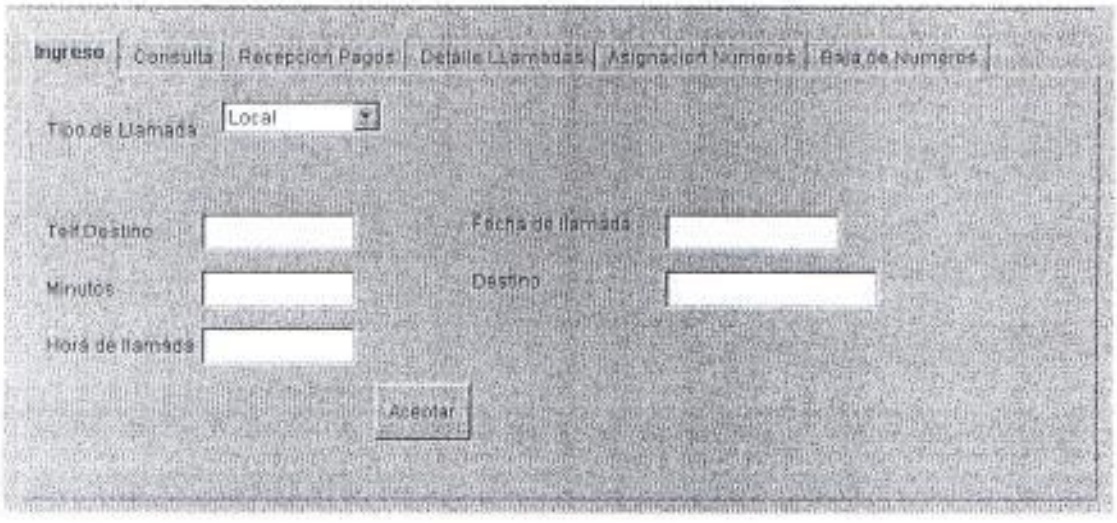



Figura : Layout de Ingreso

Layout de Consulta

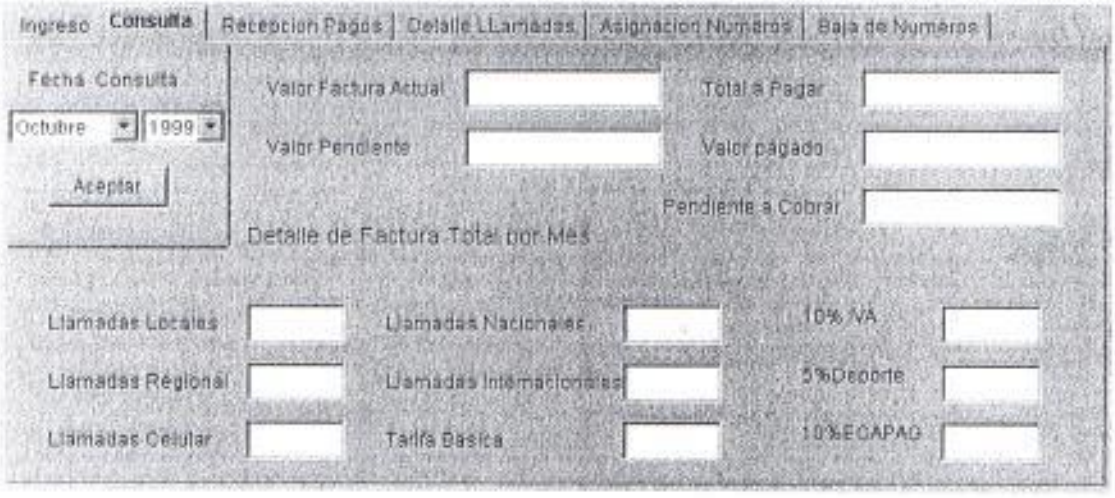



Figura : Layout de Consulta

Layout de Recepcion

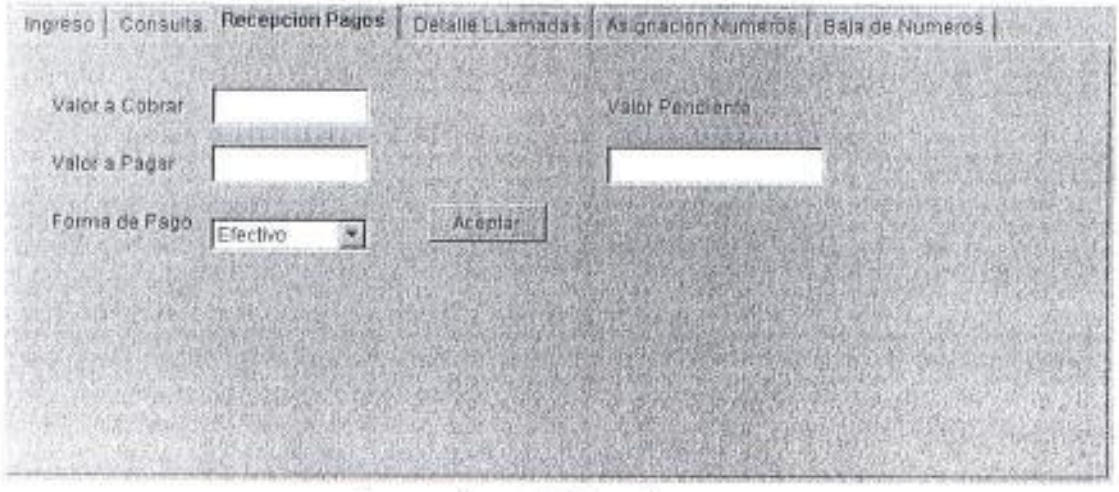



Figura : Layout de Recepcion

Layout de Detalle Llamadas

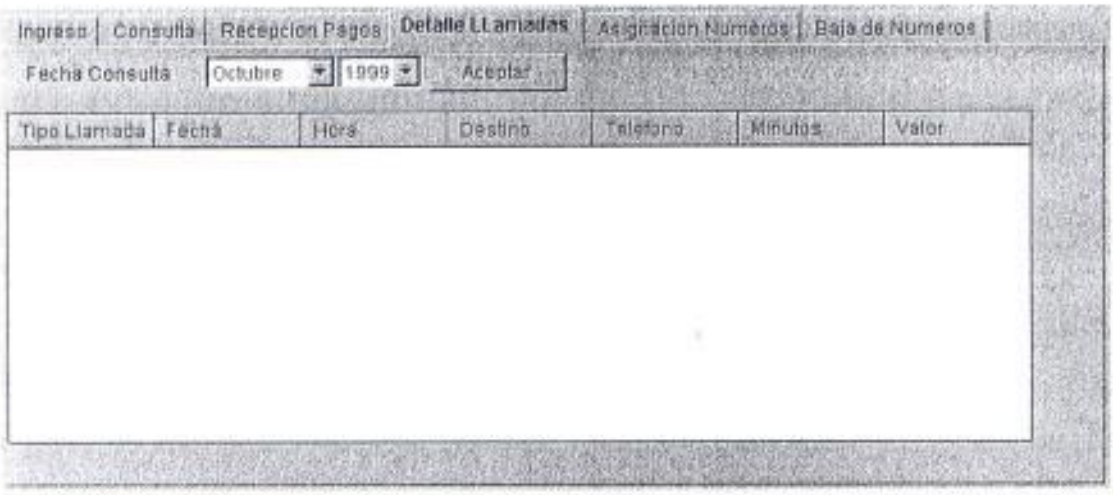



Figura : Layout de Detalle Llamadas

Layout de Asignacion Numeros


Figura : Layout de Asignacion Numeros

Layout de Baja de numeros

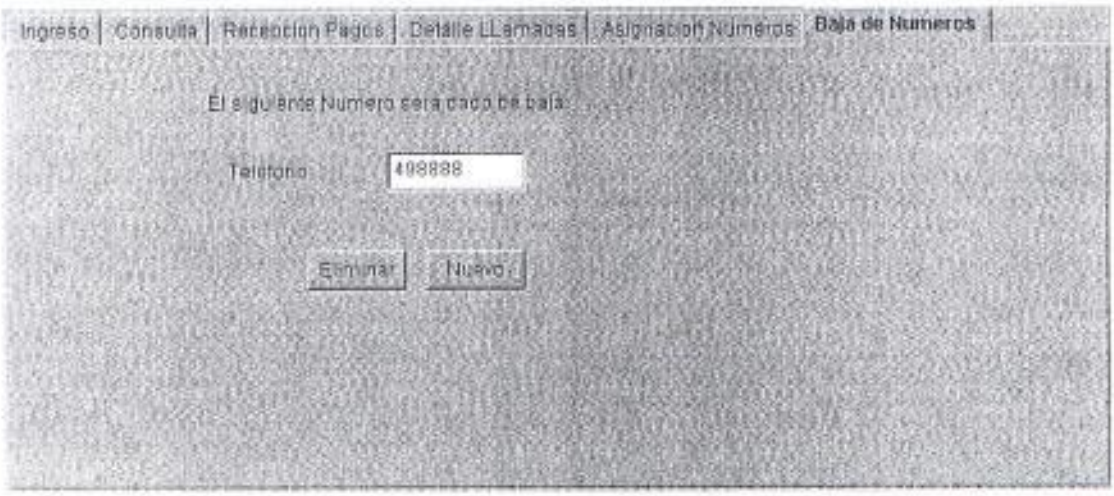



Figura : Layout de Baja de numeros

## 5.3.1.2 DIAGRAMA DE ANALISIS DE INTERACCION DE OBJETOS:

### 5.3.1.2.1 ESCENARIO VERIFICAR NUMERO EN PRINCIPAL

5.3.1.2.1 VERIFICACION EXITOSA:

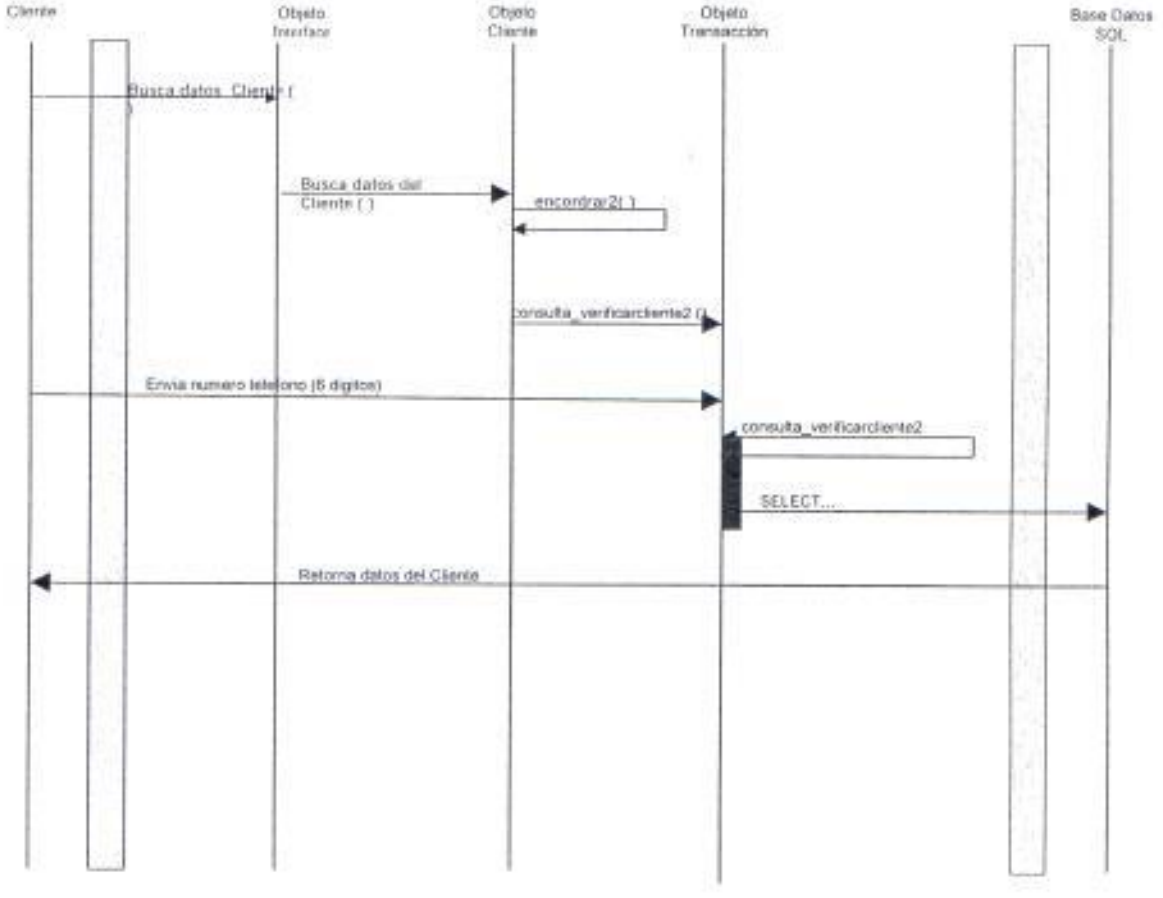



Figura : Verificacion de Numeros Exitosa

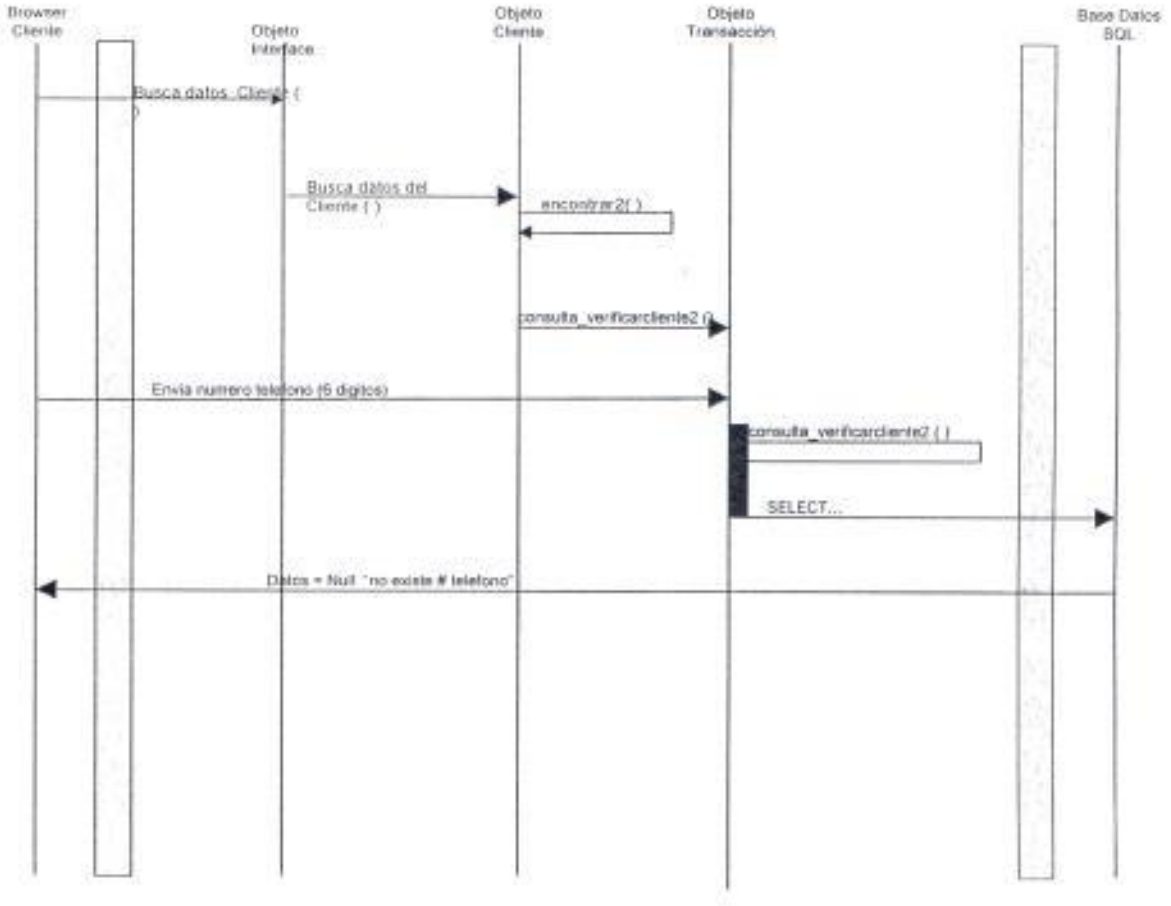



Figura : Verificacion de Numeros Fallida No existe Numero

## 5.3.1.2.2 ESCENARIO INGRESO DE CONSUMO.

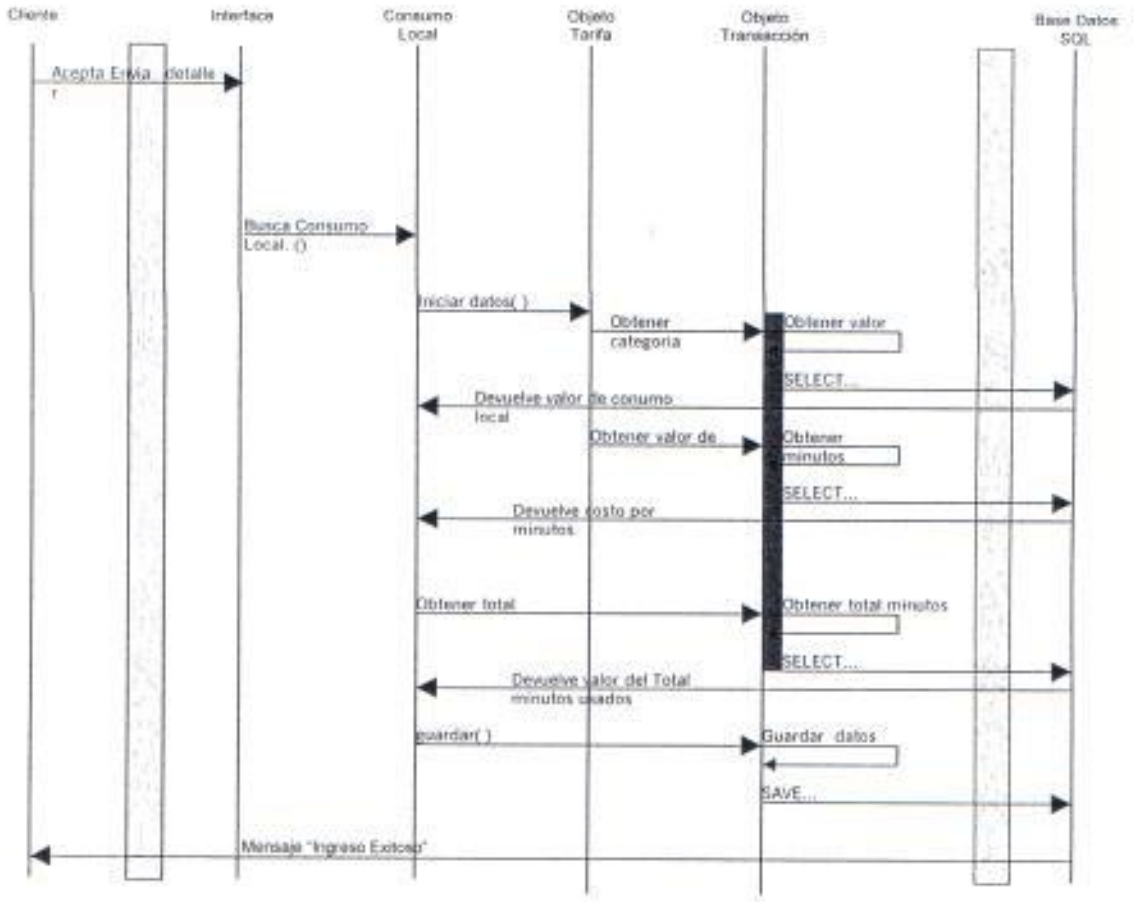



Figura : Ingreso de Consumo Exitoso Local

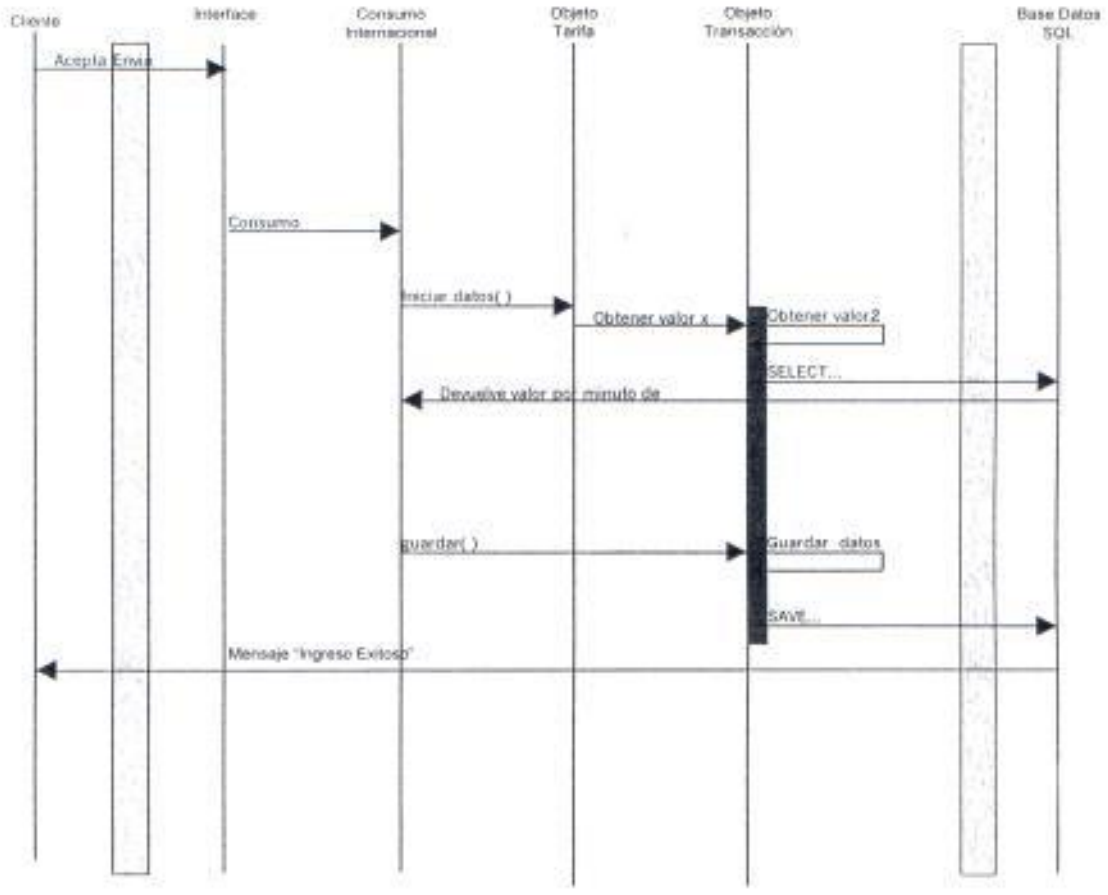



Figura : Ingreso de Consumo Exitoso Internacional

## 5.3.1.2.3 ESCENARIO CONSULTA.

5.3.1.2.3.1 CONSULTA EXITOSA.

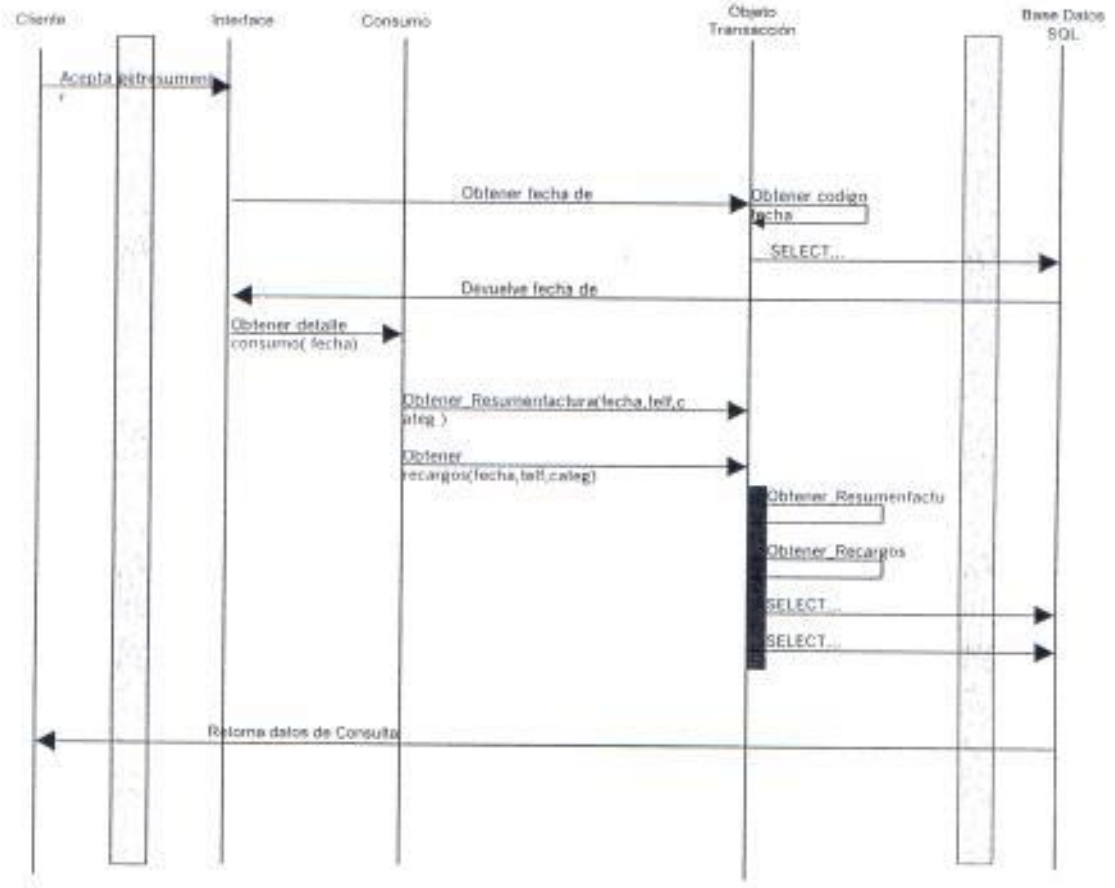



Figura : Consulta Exitosa

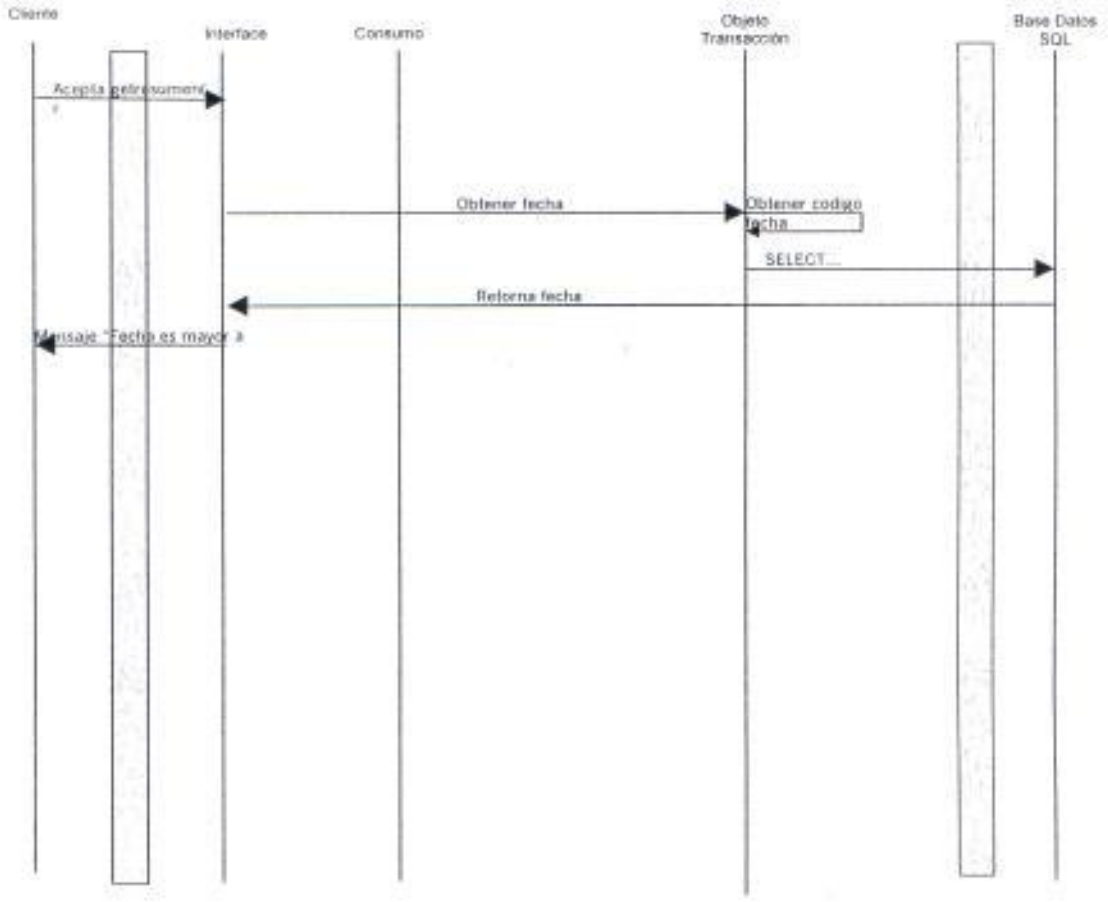



Figura : Consulta Fallida Feche Posterior

## 5.3.1.2.4 ESCENARIO RECEPCION DE PAGOS.

RECEPCION EXITOSA

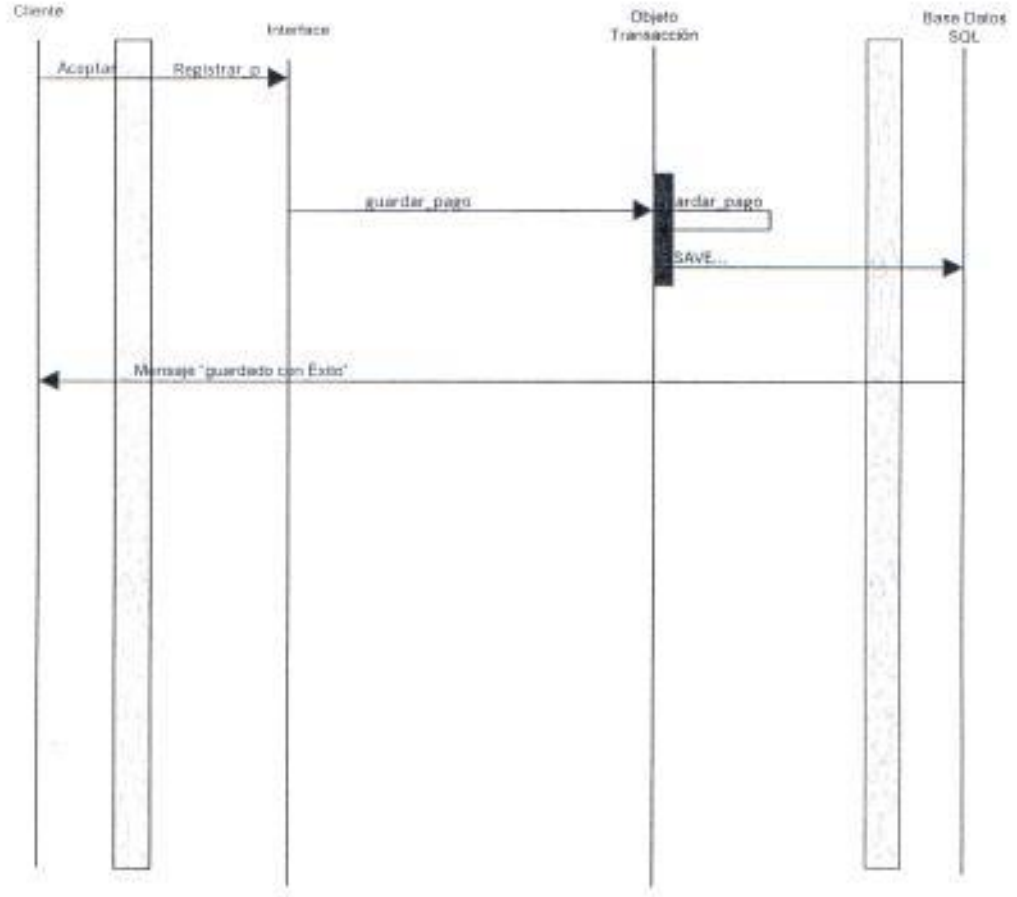



Figura : Recepcion de Pagos Exitosa

## 5.3.1.2.5 ESCENARIO DETALLE DE LLAMADAS.

5.3.1.2.5.1 DETALLE DE LLAMADAS EXITOSO

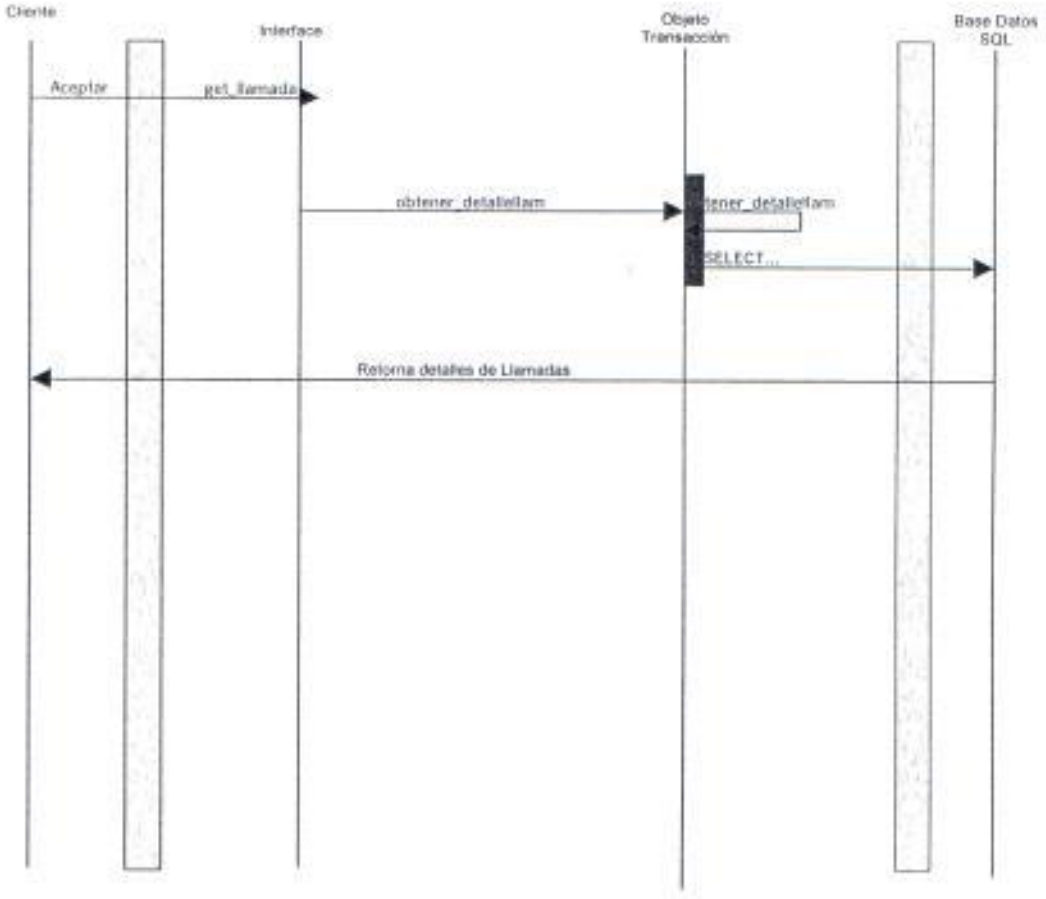



Figura : Detalle de Llamadas Exitoso

## 5.3.1.2.6 ESCENARIO ASIGNACION DE NUMEROS.

5.3.1.2.6 ASIGNACION DE NUMEROS EXITOSA

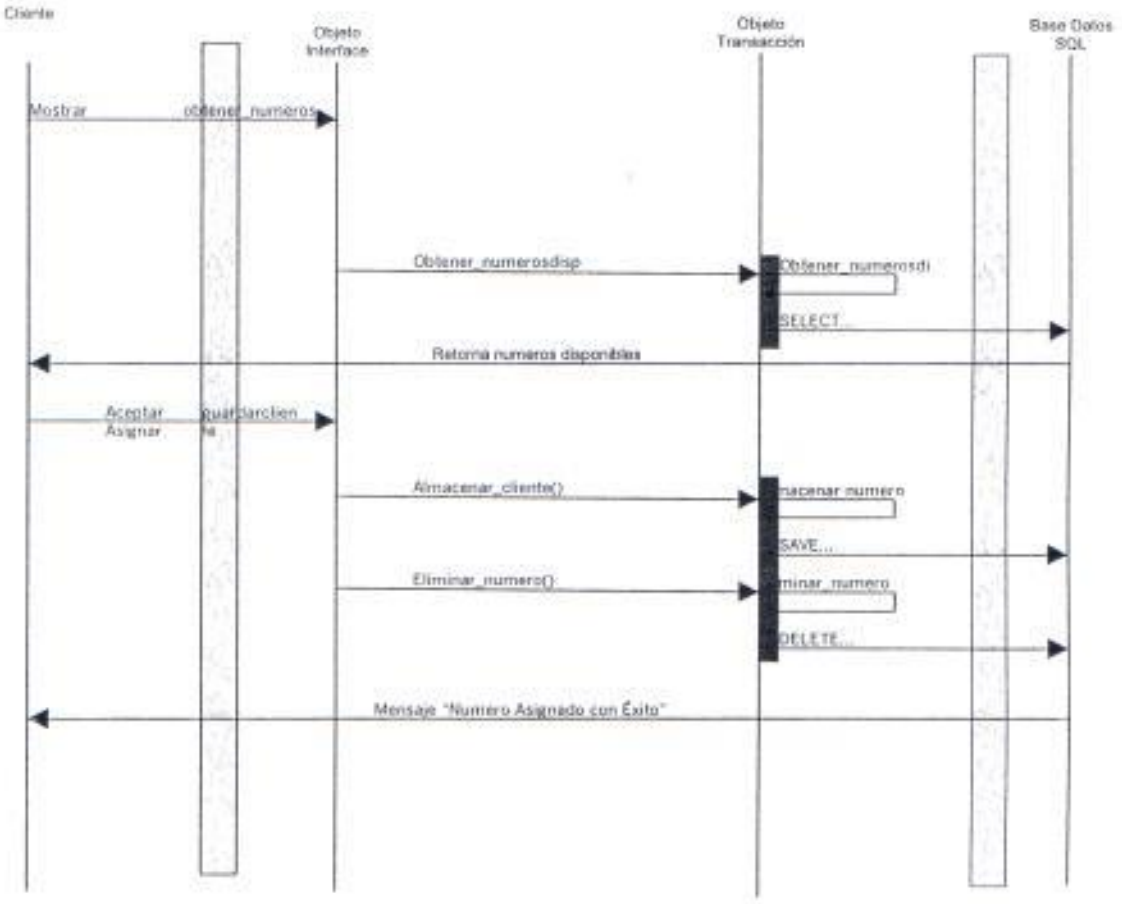



Figura : Asignacion de Numeros Exitosa

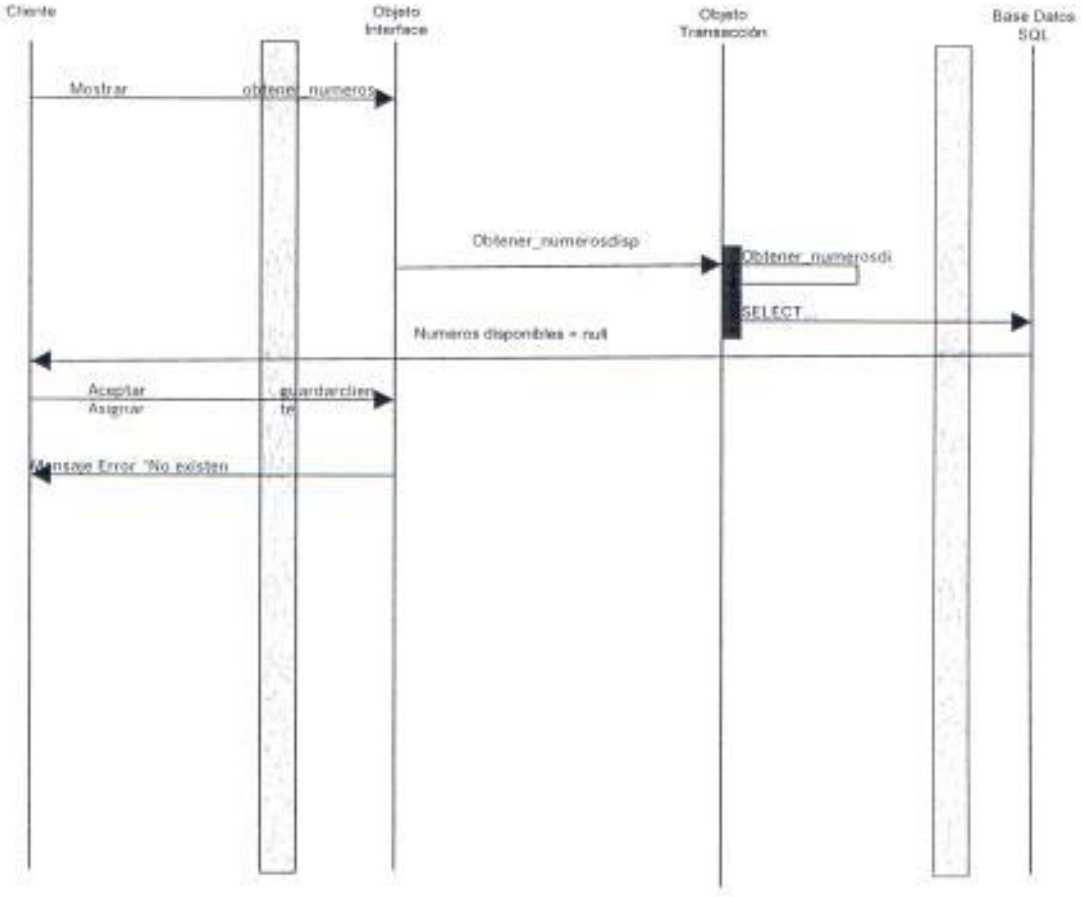



Figura : Asignacion de Numeros Fallida No existen Disponibles

## 5.3.1.2.7 ESCENARIO BAJA DE NUMEROS.

5.3.1.2.7.1 BAJA DE NUMEROS EXITOSO

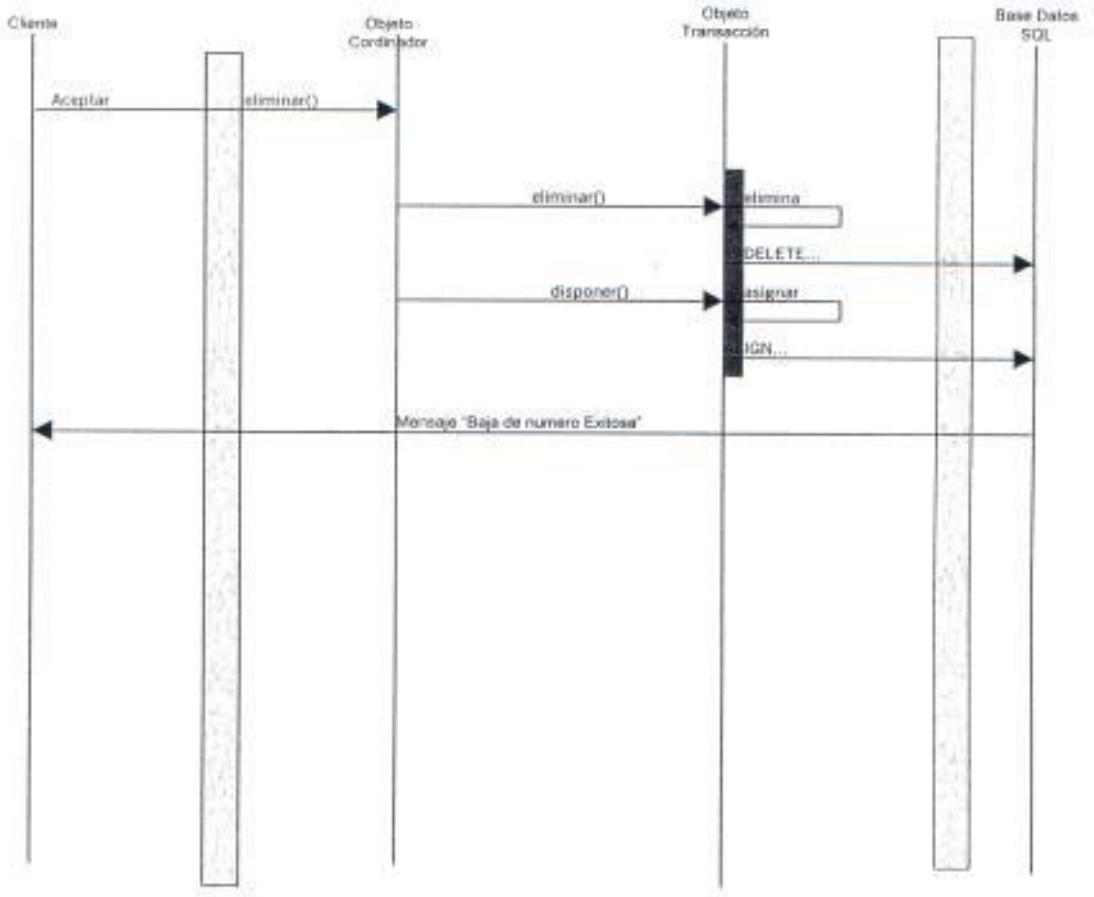



Figura : Baja de Numeros Exitosa.

## 5.3.1.3 DIAGRAMA DE DISEÑO DE INTERACCION DE OBJETOS:

## 5.3.1.3.1 ESCENARIO VERIFICAR NUMERO EN PRINCIPAL

VERIFICACION EXITOSA

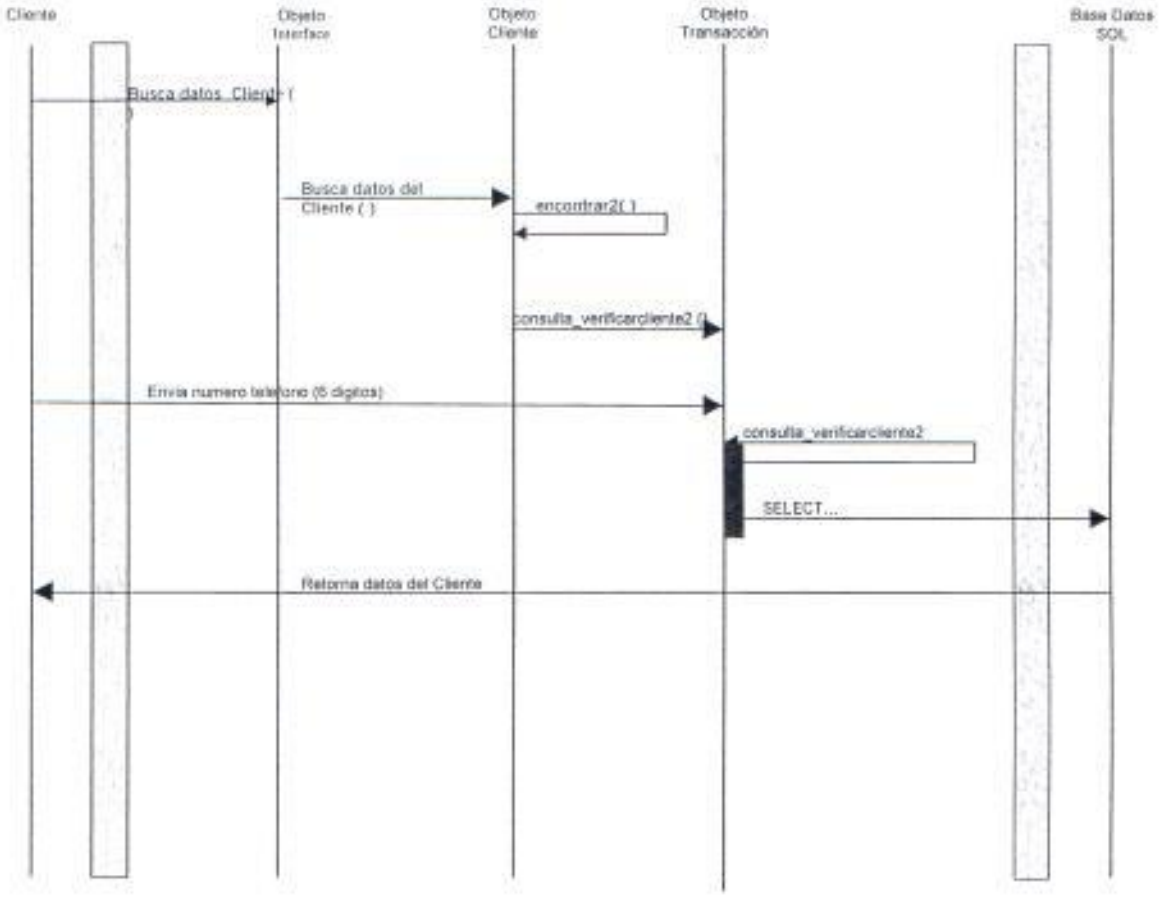



Figura : Verificacion de Numeros Exitosa

## 5.3.1.3.2 ESCENARIO INGRESO DE CONSUMO.

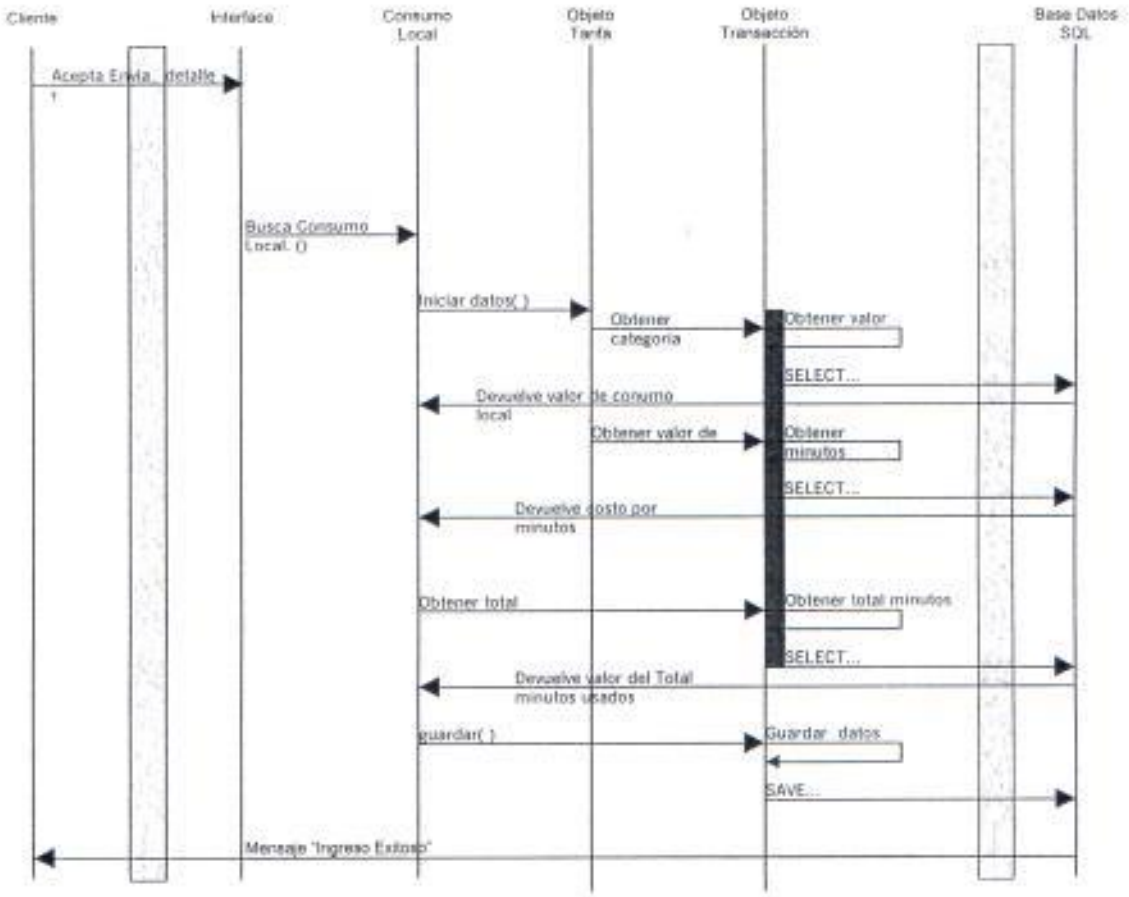



Figura : Ingreso de Consumo Exitoso Local

## 5.3.1.3.3 ESCENARIO INGRESO DE CONSUMO.

INGRESO INTERNACIONAL EXITOSO

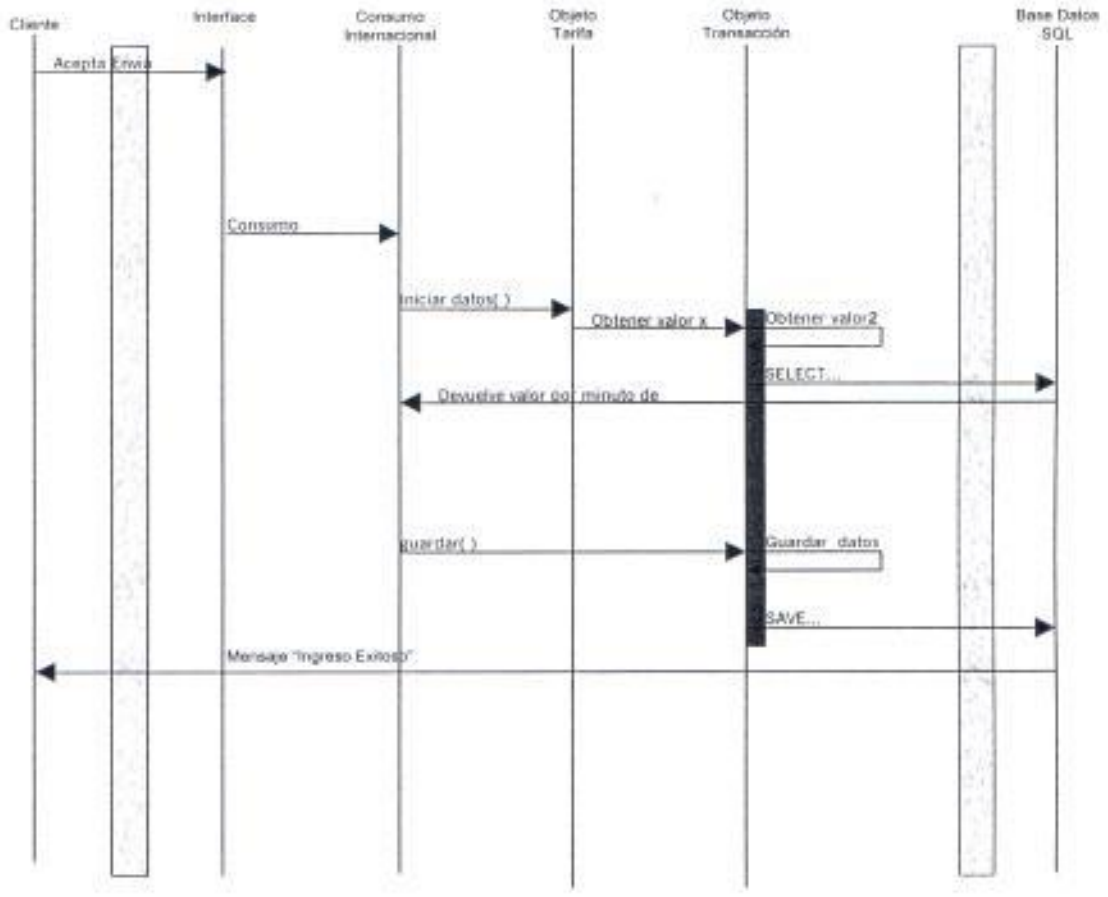



Figura : Ingreso de Consumo Exitoso Internacional

## 5.3.1.3.4 ESCENARIO CONSULTA.

CONSULTA EXITOSA.

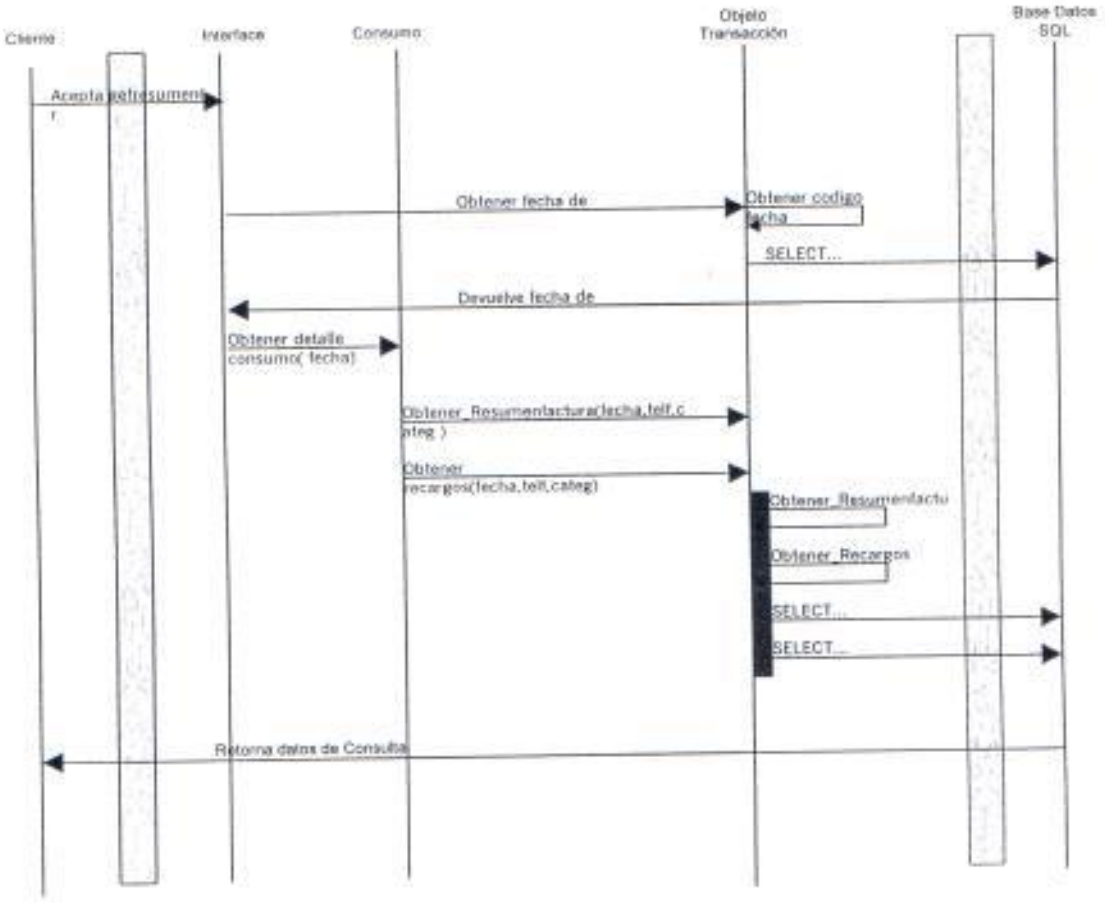



Figura : Consulta Exitosa

## 5.3.1.3.5 ESCENARIO RECEPCION DE PAGOS.

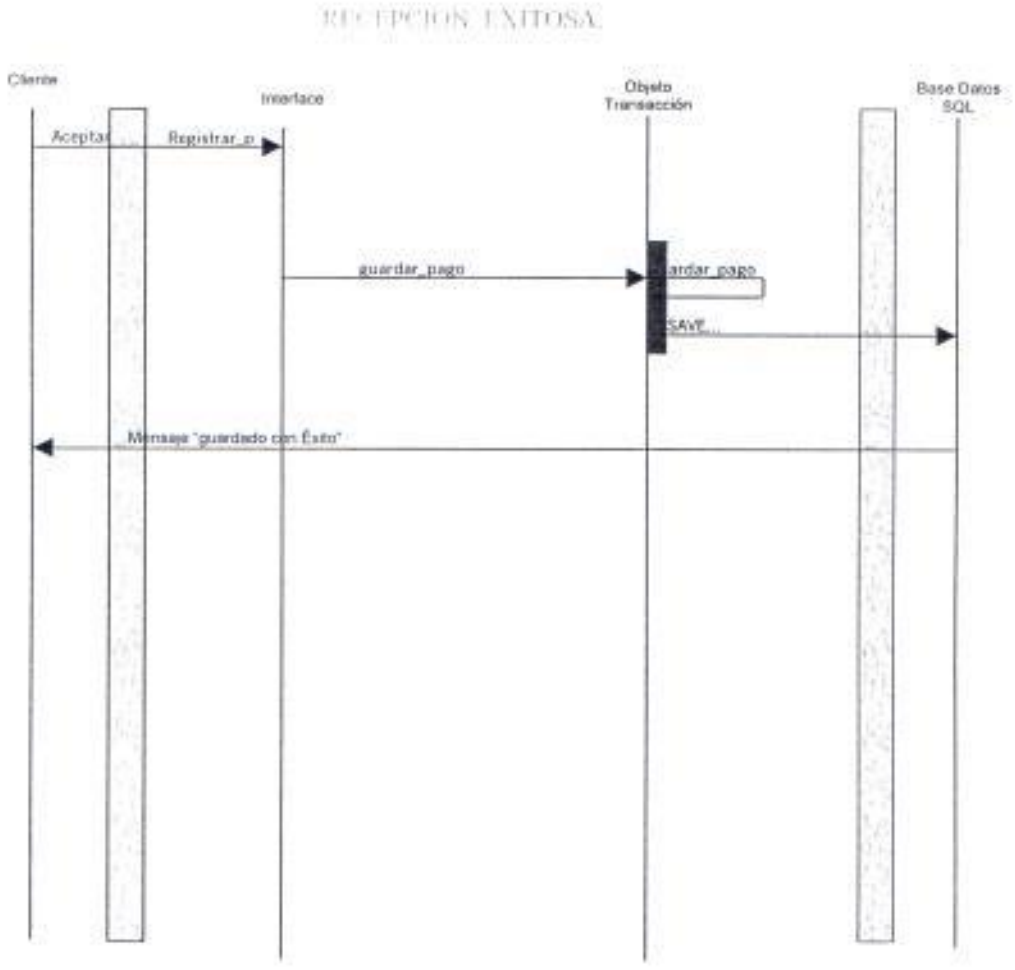



Figura : Recepcion de Pagos Exitosa

## 5.3.1.3.6 ESCENARIO DETALLE DE LLAMADAS.

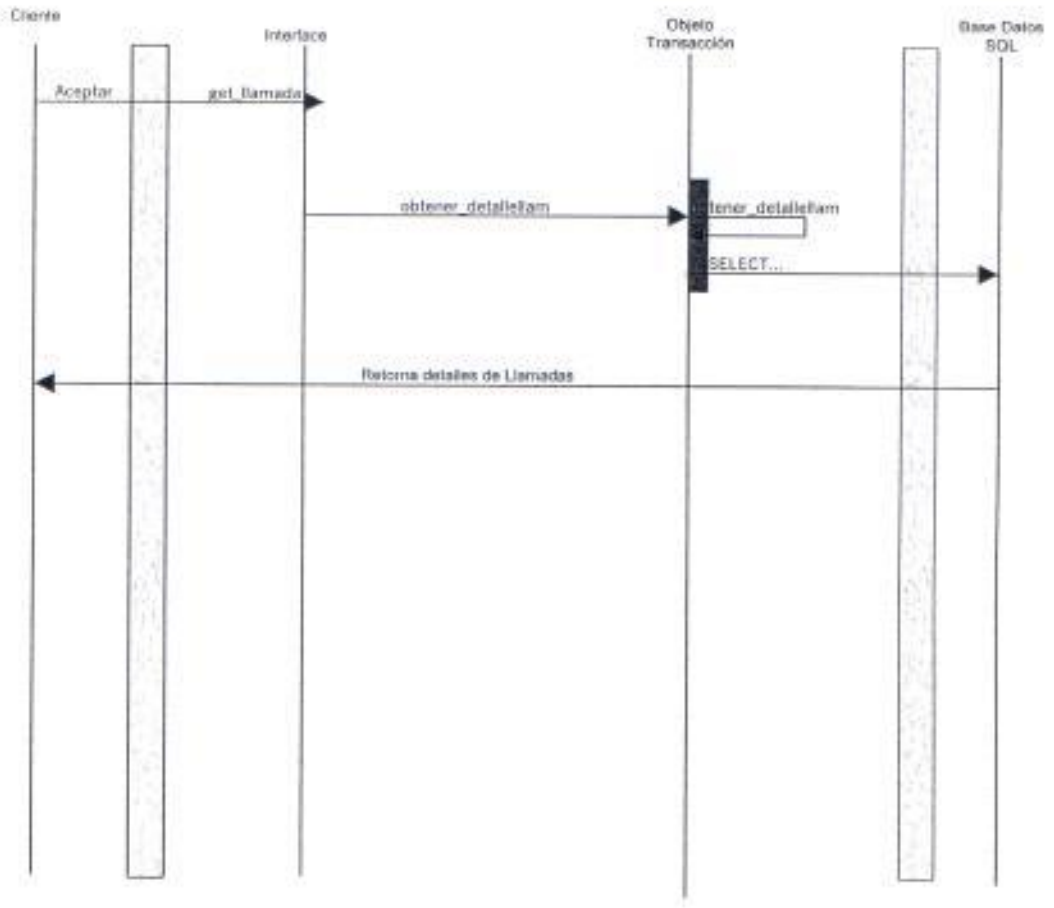



Figura : Detalle de Llamadas Exitoso

## 5.3.1.3.7 ESCENARIO ASIGNACION DE NUMEROS.

ASIGNACION DE NUMEROS EXITOSA.

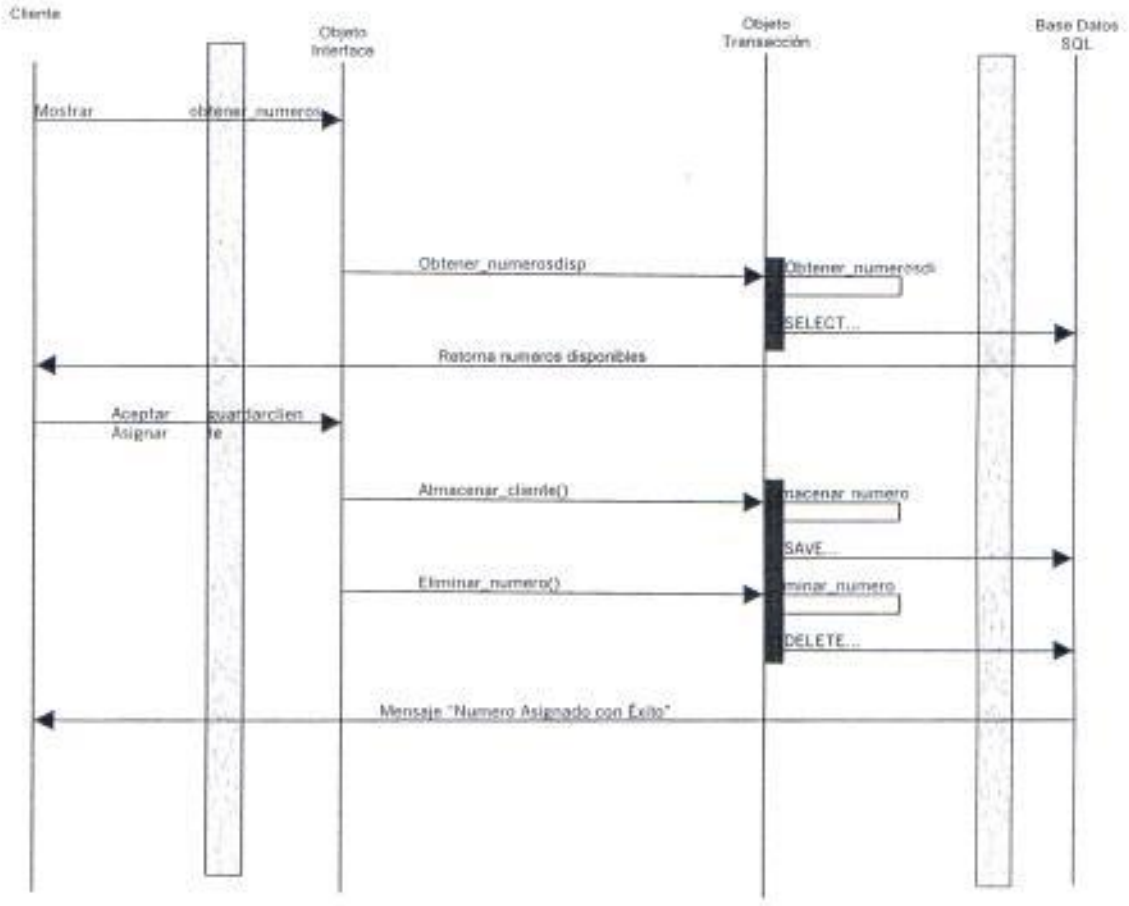



Figura : Asignacion de Numeros Exitosa

## 5.3.1.3.8 ESCENARIO BAJA DE NUMEROS.

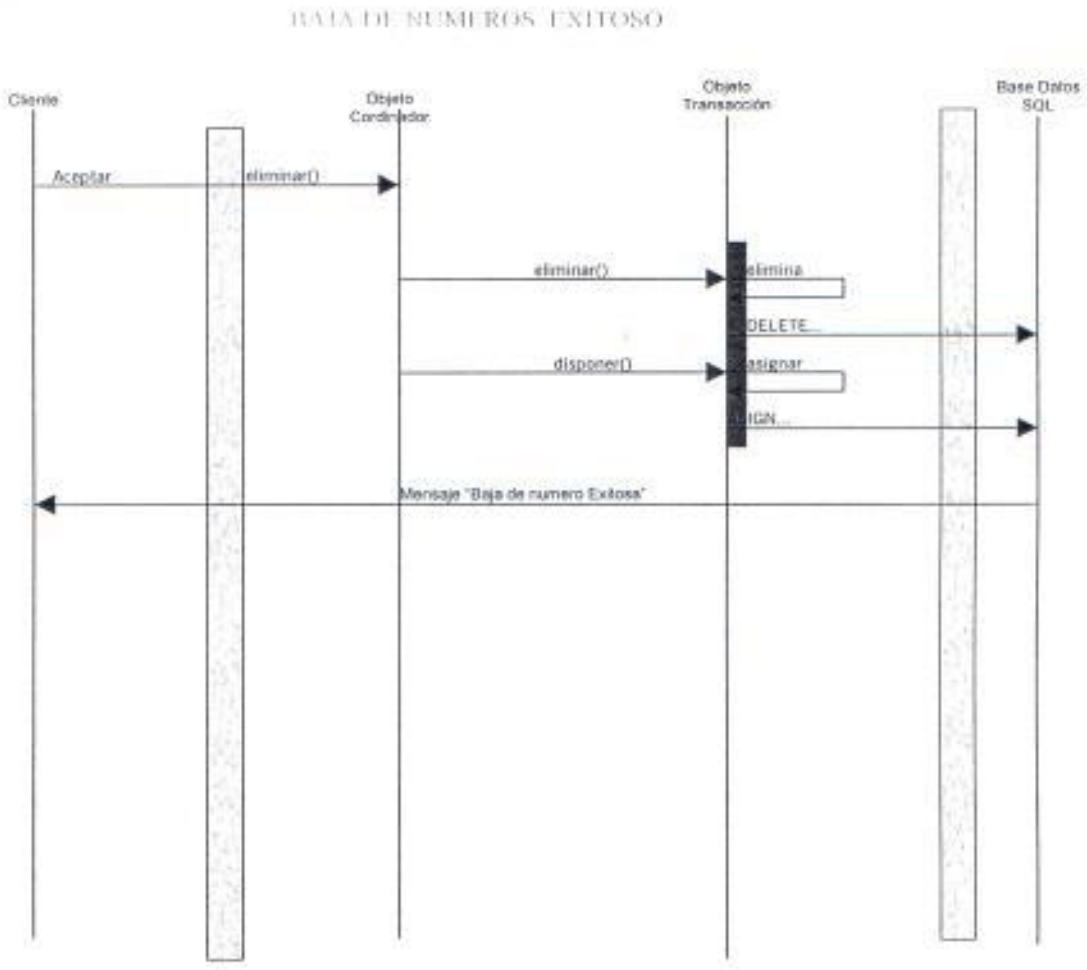



Figura : Baja de Numeros Exitosa.

### 5.3.2 Modelo Estatico

#### 5.3.2.1 Modelo de Analisis de Objetos.

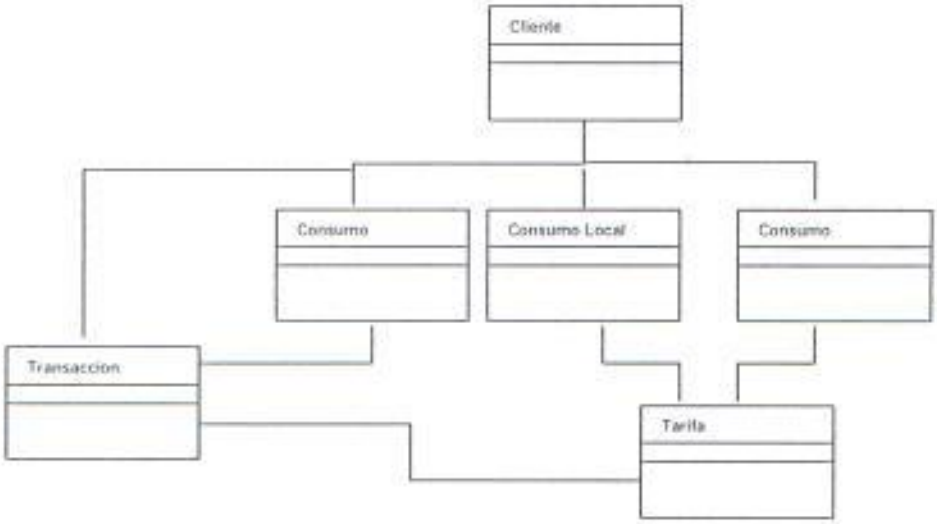



Figura : Análisis de Objetos

#### 5.3.2.2 Modelo de Diseño de Objetos.

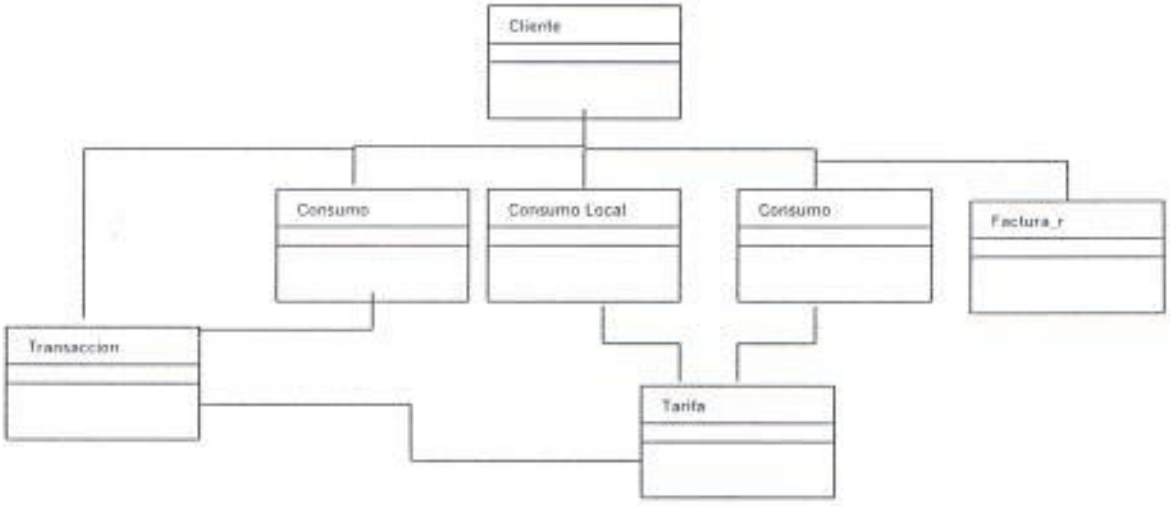



Figura : Diseño de Objetos

## 5.4 Modelo de Implementación

Una vez hecho el diseño de objetos se procede a hacer una definición de las clases y de las relaciones desarrolladas.

Esta definición se la realiza por medio de un lenguaje cualquiera, en nuestro caso utilizamos el lenguaje Java.

### 5.4.1 Diseño Descripcion de Clases.

Las siguientes clases son descritas y luego desarrolladas para finalmente ser ubicadas como objetos en el servidor para luego ser llamadas por la interface de Corba. Se describe los métodos que esta clase tendrá..

#### 5.4.1.1 Clase Cliente

Es la encargada de llevar a cabo todas las acciones que se refieren a un Cliente . Esta clase invocará a la clase Transaccion.

Realiza 2 tipos de transacciones. Primero trae los datos del cliente de acuerdo al número de teléfono que se ingrese. Segundo permite dar de Baja a un número.

Métodos:

- **Cliente:** Es el constructor de la clase, encargado de inicializar las variables y también actualiza la referencia a la clase Transacción que llega como parámetro .

  Parámetros de entrada:

  rq, (tipo Transacción): instancia de la clase Transacción.

  No devuelve nada.

- **encontrar :** Retorna datos de Cliente según numero de Teléfono y Cédula. Es encargado de verificar la información de consulta de un cliente si existe registrado en el sistema.

  Parámetros de entrada:

  telf, (tipo string): el número de teléfono.

  ced, (tipo string): el número de cédula.

  Devuelve una estructura con toda la información del cliente.

- **encontrar2:** Retorna datos de Cliente según numero de Teléfono. Igual que el método anterior, pero con la diferencia de que busca información solo en base al número de teléfono, de hecho este es el método que se usa en el sistema.

  Parámetros de entrada:

  telf, (tipo string): el número de teléfono del cliente.

Devuelve una estructura que contiene la información del cliente.

- **eliminar:** Es para dar de baja un numero. Borra de la base de datos.

  Parámetros de entrada:

  teléfono,(tipo string): el número de teléfono del cliente.

  Devuelve un entero que indica condición de éxito o fracaso.

- **disponer:** Pone a disposición números telefónicos según área.Quedan disponibles los numeros que han sido eliminados en el metodo anterior.

  Parámetros de entrada:

  teléfono,(tipo string): el número de teléfono del cliente.

  área,(tipo string): el código de área del número de teléfono.

  No devuelve nada.

### 5.4.1.2 Clase Consumo

Es el objeto encargado de traer los datos de consumo del mes de facturación u otro mes anterior, de acuerdo al número ingresado en principal y de acuerdo a la fecha que se quiera consultar. Esta clase también es la clase padre de otras dos clases que heredan propiedades de esta clase. Estas son la clase Consumo_internacional y la clase Consumo_local.

Metodos:

- **Consumo:** Este es el método constructor y es el método encargado de la inicialización de las variables. Este método además recibe como parámetros el número de teléfono, la categoría, la fecha de facturación y al igual que las otras clases la referencia a la clase Transacción.

  Parámetros de entrada:

  fecha_facturacion, (tipo entero): El código de la fecha de facturación.

  telf, (tipo string): El número de teléfono del cliente.

  categoria, (tipo string): La categoría a la que pertenece el cliente.

rq, (tipo Transacción): Instancia de la clase Transacción.

No devuelve nada.

- **guardar datos:** Es para hacer un ingreso de consumo, guarda los datos de las llamadas, después de haber sido determinado el valor del consumo. Es importante señalar que este es un método general que será invocado por métodos similares en las clases Consumo_local y Consumo_internacional pero que determinan el tratamiento de los datos dependiendo del tipo de llamada realizada.

  Parámetros de entrada:

  fe,(tipo string): La fecha de la llamada.

  destino,(tipo string): El lugar de destino.

  telf_destino,(tipo string): El teléfono de destino.

  minutos,(tipo entero): Los minutos llamados.

  hora,(tipo string): La hora de la llamada.

  valor,(tipo entero): El valor de la llamada.

  pp,(tipo entero): El tipo de llamada (local, nacional, etc.).

  Devuelve un valor tipo entero que indica una condición de éxito o fracaso.

- **Obtener total llamadas:** devuelve el total de llamadas de acuerdo al tipo de llamada. Recibe como parámetro el tipo de llamada que se ha realizado, como resultado devuelve el total de las llamadas por tipo 0 local,1 regional,2 nacional,3 celular,4 internacional.

  Parámetros de entrada:

tipo, (tipo entero): Indica el tipo de llamada ya sea local, regional, etc. Devuelve el total de las llamadas en caso de éxito, caso contrario -1 .

- **Obtener total recargos:** Calcula el total de recargos en el mes. Se obtiene el total en recargos según la fecha de facturación, el parámetro que se le envía es la fecha de facturación.

  Parámetros de entrada: (ninguno).

  Devuelve total en recargos en caso de éxito, caso contrario -1 .

- **Obtener valor pendiente:** Retorna el valor pendiente. Obtiene el valor pendiente de pagar basándose en el mes anterior al de facturación.

  Parámetros de entrada: (ninguno).

  Devuelve el valor pendiente en caso de éxito, caso contrario -1.

- **Obtener detalle consumo:** Se realiza por medio de 2 consultas obteniendo el total de recargos y resumen mes de factura. Obtiene el detalle de consumo según la fecha que recibe de parámetro, coge los datos del sistema de almacenamiento de datos, no hace cálculos.

  Parámetros de entrada:

  fe, (tipo string): Contiene la fecha a consultar.

  Devuelve el resultado como una estructura

### 5.4.1.3 Clase Consumo Internacional

Esta clase proveerá al sistema con los métodos necesarios para determinar el consumo de una llamada internacional . Esta clase hereda métodos y propiedades de su clase padre que es la clase Consumo. Una clase a la que invocará es la clase Tarifa.

Métodos:

- **Consumo_internacional:** Este es el método constructor y está encargado de la inicialización de los parámetros además realizará una invocación al constructor de su clase padre, que es la clase Consumo, al que le pasará los parámetros de inicio. El constructor además hará la inicialización de la instancia a la clase Tarifa.

  Parámetros de entrada:

  fecha_facturacion, (tipo entero): El código de la fecha de facturación.

  numero,(tipo string): El número de teléfono.

  catego, (tipo string): La categoría a la que pertenece el cliente.

  kl, (tipo Transacción): La instancia de la clase Transacción.

  No devuelve nada.

- **iniciar datos:** obtiene el valor de las llamadas internacionales según el destino. Está encargado de obtener valores de tarifa, obtenidos de la referencia a la clase Tarifa. Estos resultados serán usados para el cálculo del consumo en el método guardar de la clase.

  Parámetros de entrada: (ninguno).

  Devuelve un valor tipo entero que indica condición de éxito o fracaso.

- **Guardar:** Guarda en Base de Datos el consumo de llamada internacional. Por medio de este método se guardará el valor calculado después de realizar los cálculos necesarios, hace referencia al método equivalente en la clase padre.

  Parámetros de entrada:

  fechas, (tipo string): La fecha en que se realizó la llamada.

  tminutos, (tipo entero): Los minutos llamados.

  telfs, (tipo string): El número de teléfono hacia el que se realizó la llamada.

  horas, (tipo string): La hora en la que se realizó la llamada.

  destis, (tipo string): El lugar de destino de la llamada.

  c, (tipo entero): Esta variable indica el tipo de destino internacional, valor basado en la tabla de destinos internacionales.

  Devuelve valor tipo entero indicando condición de éxito o fracaso.

#### 5.4.1.4 Clase Consumo Local

Se la utiliza para realizar operaciones relacionadas con un tipo de consumo local. Ya sea que se lo escoja como Local, Regional, Nacional o Celular. Esta clase es muy parecida a la clase anterior, pero con la diferencia de que lleva a cabo tareas específicas de esta clase. Al igual que la clase Consumo_internacional esta clase también hereda las propiedades y métodos de la clase padre que es la clase Consumo.

Metodos:

- **Consumo_local:** Este es el constructor de la clase y también inicializará las variables necesarias y creará una instancia a la clase Tarifa.

  Parámetros de entrada:

  fecha_t, (tipo entero): El código de la fecha de facturación.

  tel_,(tipo string): El número de teléfono del cliente.

  cate_,(tipo string): La categoría del cliente.

  tg, (tipo Transaccion): Instancia de la clase Transacción.

  No devuelve nada.

- **iniciar datos:** obtiene el valor de las llamadas locales, obtiene minutos gratis y total de minutos llamados. Muy parecido al método iniciar de la clase Consumo_internacional, en este se obtienen valores de las llamadas locales después de haber hecho la referencia a la clase Tarifa.

  Parámetros de entrada: (ninguno).

  Devuelve un valor entero indicando una condición de éxito o fracaso.

- **guardar:** Guarda en Base de Datos el consumo de llamada Local. Se guarda el valor calculado después de hacer los cálculos necesarios, también trabaja con el método guardar de la clase padre.

  public int guardar(String fec,int minut,String telf,String hora,int tip,String dz)

  Parámetros de entrada:

  fec, (tipo string): La fecha de la llamada.

  minut, (tipo entero): La cantidad de minutos llamados.

  telf, (tipo string): El número de teléfono hacia el que se hizo la llamada.

  hora, (tipo string): La hora en que se realizó la llamada.

  tip, (tipo entero): El código del tipo de llamada ya sea esta local, regional, etc.

  dz, (tipo string): El lugar de destino de la llamada.

### 5.4.1.5 Clase Tarifa

Es accesada por los objetos Consumo Intenacional y Consumo Local, para poder realizar el calculo del costo de cada llamada de acuerdo al tipo de llamada, la localizacion y su tarifa. Esta clase hace como de tabla de tarifas para la clase Consumo, Consumo_local y Consumo_internacional .Contiene métodos que harán referencia a la clase Transacción para obtener datos del sistema de almacenamiento de datos.

Metodos:

- **Tarifa:** Este es el constructor de la clase, está encargado de la inicialización de las variables y de obtener la referencia de la instancia a la clase Transacción.

  Parámetros de entrada:

  fecha,(tipo entero): Este es el código de la fecha de facturación.

  rq, (tipo Transacción): Esta es la instancia de la clase Transacción.

  No devuelve nada.

- **obtener minutos:** Obtiene los minutos gratis de una llamada local, ssegún la categoría y la fecha.

  Parámetros de entrada:

cate,(tipo string): la categoría a la que pertenece el cliente

Devuelve un valor de tipo entero que contiene los minutos gratis en caso de tener éxito y un -1 en caso contrario.

- **Obtener valor:** obtiene el valor de una llamada local, regional, nacional, celular.

  public int obtener_valor(int tipo,String cat)

  Parámetros de entrada:

  tipo, (tipo entero): El código del tipo de llamada.

  cat, (tipo string): La categoría a la que pertenece el cliente.

  Devuelve un valor de tipo entero que contiene el valor de la llamada en caso de tener éxito, caso contrario devuelve un valor de -1.

- **Obtener valor2**: obtine el valor de una llamada Internacional. Por medio de este método se obtiene el valor de una llamada según la fecha y el destino.

  Parámetros de entrada:

  des, (tipo entero): Contiene el código de destino de una llamada según la zona a la que pertenece el lugar hacia el que se llama, estos valores están codificados en la base de datos.

  Devuelve un valor de tipo entero que contiene el valor de la llamada en caso de tener éxito, caso contrario devuelve un valor de -1.

### 5.4.1.6 Clase Factura_r

Es un objeto que sirve para el refrescamiento de los datos. Se lo llama en la invocacion a la funcion FacturacionImpl. Esta clase estará encargada de mantener información importante para el sistema de facturación, información que permitirá mantener al sistema con la información actualizada. Esta clase invoca a la clase Transacción y a la clase Consumo.

Métodos:

- **Factura_r:** Este es el método constructor y es el encargado de la inicialización de las variables. Obtiene la instancia de la clase Transacción.

  Parámetros de entrada:

  rq, (tipo Transacción): instancia de la clase Transacción.

  No devuelve nada.

- **obtener mes de facturacion:** Obtiene el mes que se esta facturando. Este método se encarga de obtener el código del mes que está siendo facturado, es decir el mes que está registrado como activo en el sistema de almacenamiento de datos.

  Pararametros de entrada: (ninguno).

Devuelve un valor de tipo entero que es el código del mes de facturación.

- **Insertar mes de facturacion:** registra como activo al mes de facturacion. Este método está encargado de registrar el nuevo mes de facturación, es decir registrar al mes que va a hacer de fecha de facturación como activo.

  Parámetros de entrada:

  fecha, (tipo String): Contiene el mes nuevo de facturación.

  Devuelve un valor de tipo entero que indica con 0 o -1 si la operación fue llevada con éxito o fracaso (Todos los métodos que devuelven este tipo de indicativo de éxito o fracaso usan el 0 como éxito y el -1 como fracaso).

- **Iniciar detalle de factura:** genera informacion respecto al mes de facturacion. Este método trabaja con la tabla detalle_factura, con este método lo que se hace es generar la información con respecto al mes de facturación que recién se ha asignado.

  Parámetros de entrada:

  fecha, (tipo entero): Contiene el código de l mes de facturación.

  teléfono, (tipo String): Contiene el némero de teléfono con el que se va a generar la información.

categoria, (tipo string): Contiene la categoria a la que pertenece el usuario.

inserción, (tipo entero): un código para saber si se está iniciando esta función por primera vez o por segunda vez (criterio definido por el programador).

Devuelve un valor de tipo entero indicando si la operación fue llevada con éxito o no.

#### 5.4.1.7 Clase Transaccion

Este es el objeto encargado de transformar las peticiones del cliente a travez de los demas objetos en sentencias SQL y de esta manera poder interactuar con la base de Datos.

Esta es la clase que maneja la mayoría de la comunicación con el sistema de persistencia de datos. En nuestro caso el sistema de persistencia de datos es una base de datos SQL Server, pero debido a que todas las transacciones con este están encapsuladas en esta clase, será muy fácil ajustar todos los métodos de esta clase si se decide cambiar el sistema de persistencia de datos.

Todas las clases hacen uso de esta clase, invocando los métodos que esta contiene. Por obvias razones para facilitar la interacción de las demás clases, la clase Transacción posee muchos métodos.

Métodos:

Son los métodos utilizados por todos los objetos o transacciones que requieran el uso de la base de datos.

- **Transacción.-** Este es el constructor de la clase y en este caso no inicializa nada ya que al instanciar la clase automáticamente se inicializan las variables que interactuarán con el sistema de almacenamiento de datos.

  Parámetros de entrada: (ninguno).

  No devuelve nada.

- **open:** Tiene a cargo iniciar la conexión con la base datos por lo que iniciará las variables necesarias.

  Parámetros de entrada: (ninguno).

  No devuelve nada.

- **closeconeccion:** Cierra la conección con la base de datos.

  Parámetros de entrada: (ninguno).

  No devuelve nada.

- **guardarllamada:** Este método guardará una llamada ingresandola en el sistema de almacenamiento de datos.

  Parámetros de entrada:

  fe,(tipo string): La fecha en que se realizó la llamada.

destino, (tipo string): El lugar de destino hacia el que se llamó.

telf_destino, (tipo string): El teléfono de destino hacia el que se llamó.

minutos,(tipo entero): Los minutos que duró la llamada.

hora,(tipo string): La hora en que fue realizada la llamada.

valor,(tipo entero): El valor de la llamada.

fecha,(tipo entero): El código de la fecha de facturación.

telf,(tipo string): El número de teléfono del cliente.

pp, (tipo entero): El código de ltipo de llamada.

Devuelve un valor de tipo entero indicando una condición de éxito o fracaso.

- **obtener_totalllamadas:** Por medio de este método se obtiene el total de las llamadas realizadas por algún usuario, según el tipo de llamada que será pasado como un parámetro a la función.

  Parámetros de entrada:

  tipo,(tipo entero): Contiene el código del tipo de llamada (local, nacional, etc.).

  telef, (tipo string): Contiene el número de teléfono en base al que se va a realizar la consulta.

  fecha,(tipo entero): Contiene el código de la fecha del mes que está haciendo de fecha de facturación.

  Devuelve un valor de tipo entero que es valor total de las llamadas.

- **obtener_totalrecargos:** Este un método parecido al anterior, pero con la diferencia de que obtiene el total de los recargos que se le hacen a una llamada telefónica.

  Parámetros de entrada:

  fecha,(tipo entero): Contiene el código de la fecha de facturación.

  Devuelve un valor de tipo entero, que es el valor total de los recargos.

- **obtener_valorpendiente:** Obtiene el valor pendiente de pagar de llamadas realizadas en el mes anterior.

  Parámetros de entrada:

  fecha,(tipo entero): Contiene el código de la fecha de facturación.

  telef,(tipo string): Contiene el número de teléfono en base al cual se va a realizar la consulta.

  Devuelve un valor de tipo entero que es el valor pendiente.

- **obtener_resumenfactura:** Obtiene de la base de datos el detalle de los datos que se mostrarán en el mes que se está consultando el detalle de la factura. Devuelve un arreglo con todos los datos obtenidos.

  Parámetros de entrada:

  fecha, (tipo string): Contiene la fecha en base a la cual se va a realizar la consulta.

telef, (tipo string): El número de teléfono en base al cual se va a realizar la consulta.

cate, (tipo string): La categoría a la que pertenece el cliente (viene como letra A, B, etc.).

Devuelve un arreglo de tipo string, en el cual van los datos de un mes a consultar.

- **obtener_recargos:** Obtiene los recargos detallados desde la base de datos, según el código de fecha enviado.

  Parámetros de entrada:

  fecha, (tipo entero): Contiene el código de la fecha de facturación.

  tel, (tipo string): Contiene el número de teléfono a consultar.

  cate, (tipo string): Contiene la categoría a la que pertenece el cliente.

  Devuelve un arreglo de tipo string que contiene los resultados de los datos a consultar.

- **obtener_valor:** Obtiene el valor del minuto de una llamada local desde la base de datos.

  Parámetros de entrada:

  tip, (tipo entero): Contiene el código del tipo de llamada.

  cate, (tipo string): Contiene la categoría a la que pertenece el cliente.

fech, (tipo entero): Contiene el código de la fecha de facturación.

Devuelve un valor de tipo entero que es valor a consultar.

- **obtener_valor2:** Obtiene el valor del minuto de una llamada según el código de destino que se requiera para una llamada de larga distancia.

  Parámetros de entrada:

  public int obtener_valor2(int nu,int fech,int des) throws Exception

  nu, (tipo entero): El código del tipo de llamada.

  fech,(tipo entero): El código de la fecha de facturación.

  des, (tipo entero): Es el código del destino al que se llamó, valor obtenido de una tabla de destinos.

  Devuelve un valor de tipo entero cuyo contenido es el valor esperado.

- **obtener_minutos:** Obtiene la cantidad de minutos gratis a que tiene un usuario derecho según la categoría a la que pertenezca este.

  Parámetros de entrada:

  cat, (tipo string): Es la categoría a la que pertenece el cliente.

  fech, (tipo entero): Es el código de la fecha de facturación.

  Devuelve un valor de tipo entero cuyo resultado son los minutos gratis.

- **obtener_totalminutos:** Obtiene el total de minutos llamados para llamadas de tipo local.

  Parámetros de entrada:

  telf, (tipo string): El número de teléfono del cliente.

  fech, (tipo entero): El código de la fecha de facturación.

  *Devuelve un valor de tipo entero que es el total de los minutos.*

- **consulta_verificarcliente:** Por medio de este método se buscará mediante el número de teléfono si el cliente existe en la base de datos.

  Parámetros de entrada:

  telf,(tipo string): El número de teléfono del cliente.

  cedula, (tipo string): El número de cédula del cliente.

  Devuelve como resultado una estructura, en la que van los datos del cliente.

- **consulta_verificarcliente2:** Este método tiene el mismo objetivo que el método anterior con la diferencia de que ahora la búsqueda se hará mediante el número de teléfono y el número de cédula.

  Parámetros de entrada:

  telf, (tipo string): El número de teléfono del cliente.

Devuelve como resultado una estructura, en la que van los datos del cliente.

- **eliminar:** Elimina un número de teléfono asignado a un cliente de la base de datos.

  Parámetros de entrada:

  telf, (tipo string): Es el número de teléfono del cliente.

  Retorna un valor de tipo entero que será cero si tiene éxito.

- **asignar:** Inserta el número de teléfono que se pasa como parámetro en la tabla de números disponibles de la base de datos.

  Parámetros de entrada:

  telf,(tipo string): El número de teléfono que se ha asignado al cliente.

  area, (tipo string): El código de area al que pertenece este número.

  Devuelve un valor de tipo entero que es cero si el método tiene éxito.

- **obtener_mesfacturacion:** Obtiene el código del mes que se está facturando.

  Parámetros de entrada(ninguno).

  Devuelve un valor de tipo entero que es el código del mes que se está facturando.

- **obtener_codigofecha:** Obtiene el código del mes que está recibiendo como parámetro.

  Parámetros de entrada:

  is, (tipo string): La fecha que se va a consultar.

  Devuelve un valor de tipo entero, que es el código del mes que se ha consultado.

- **guardar_detallefactura:** Inserta los totales de llamadas obtenidos en un mes de facturación nuevo y los inserta en la base de datos.

  Parámetros de entrada:

  fech, (tipo entero): El código de la fecha de facturación.

  telefono,(tipo string): El número de teléfonbo del cliente.

  n, de tipo entero: Valor entero que contiene el valor pentiente por pagar en el mes anterior.

  tc, (arreglo de tipo entero): En este arreglo van los totales por tipo de llamada.

  rc, (tipo entero): Aqui va el valor de la factura.

  Devuelve un valor de tipo entero que es un cero en caso de haberse realizado con éxito.

- **obtener_detallellamadas:** Obtiene de la base de datos las llamadas realizadas en un mes cualquiera, estas llamadas serán mostradas en detalle .

  Parámetros de entrada:

  telf, (tipo string): El número de teléfono del cliente.

  fec, (tipo string): La fecha en base a la cual se va a hacer la consulta.

  Devuelve un vector (un vector es muy parecido a un arreglo, con la diferencia de que sirve para manejar datos de longitud variable), en el que van los resultados de las llamadas en el mes en que se consultó.

- **obtener_numerosdispon:** Obtiene desde la base de datos todos los números que se encuentran disponibles para ser asignados al cliente que solicite un listado de números disponibles.

  Parámetros de entrada:

  codigoarea, (tipo string): El código de área al que pertenece el número del cliente.

  Devuelve como resultado un vector, que contiene los números disponibles obtenidos de la base de datos.

- **almacenar_cliente:** Inserta un nuevo cliente con un número de teléfono asignado dentro de la base de datos.

en, (variable tipo estructura): Dentro de la estructuras declaradas es una variable de tipo clientstruct y almacenas principalmente datos del cliente.

es, (variable tipo estructura): Dentro de las estructuras declaradas es una variable de tipo areastruct que contiene el código de area y el número de teléfono.

Devuelve valor de tipo entero que es cero si tiene éxito.

- **eliminar_numero:** Borra un número de teléfono de la tabla de números disponibles en la base de datos.

  Parámetros de entrada:

  en,(variable tipo estructura): Dentro de las estructuras declaradas es una variable de tipo areastruct que contiene el código de area y el número de teléfono.

  Devuelve valor de tipo entero que es cero si tiene éxito.

- **guardar_pago:** Registra un pago realizado por un cliente dentro de la base de datos.

  Parámetros de entrada:

  valorpa, (variable tipo estructura): Dentro de las estructuras declaradas es una variable de tipo pagostruct, en esta estructura se guardan datos referentes a un pago.

fech, (variable tipo entero): Contiene el código de la fecha de facturación.

numero, (variable tipo string): Contiene el número de teléfono del cliente.

Devuelve como resultado un valor de tipo entero, que es cero si tiene éxito.

- **actualizar_detallefactura:** Inserta dentro de la base de datos las llamadas realizadas como totales por tipo de llamada.

  Parámetros de entrada:

  fech, (tipo entero): El código de la fecha de facturación.

  telefono, (tipo string): El número de teléfono del cliente.

  tc, (arreglo tipo entero): Aqui van los totales de las llamadas por tipo de llamada.

  rc, (tipo entero): En este caso no se está usando.

  Devuelve un valor de tipo entero que es cero si la operación se lleva con éxito.

- **insertar_mesfacturacion:** determina y registra como activo un nuevo mes como mes de facturación.

  Parámetros de entrada:

  fech, (tipo string): La fecha que se va a registrar como nueva fecha de facturación.

Devuelve un valor de tipo entero que es cero si lleva con éxito.

- **obtener_tarifabasica:** Obtiene la tarifa basica para una categoria de cliente.

  Parámetros de entrada:

  fech, (tipo entero): El código de la fecha de facturación.

  cate, (tipo string): La categoría a la que pertenece el cliente.

  Devuelve un valor de tipo entero, que será el valor de la tarifa básica.

## 5.5 Tipo de Servidor y Justificación.

El tipo de servidor de base de datos usado para nuestro sistema de facturación es el de un servidor de base de datos orientado a procesamiento de transacciones, concretamente una base de datos SQL Server 7.0.

Las respuestas de este servidor de base de datos es muy buena , además que nos permite ejecutarlo en sistemas operativos como Windows95/98.
El sistema se adaptará fácilmente a cualquier tipo de servidor web.

### 5.5.1 Diseño de los datos manejados en el servidor.

El tercer Tier es donde funciona nuestro sistema de base de datos. Todos los objetos servidores Corba actúan directa o indirectamente con este. En un sistema cliente/servidor moderno el sistema de base de datos se puede ejecutar en la misma máquina donde se ejecutan nuestros objetos Corba o en una máquina diferente.

### 5.5.2 Facilidad de adaptación

Es importante decir que el proyecto esta concebido desde la idea que el sistema se lo pueda adoptar fácilmente a cualquier sistema de base de datos, sin cambiar una sola línea de código. Esto es posible debido a

que en el proceso de diseño del sistema creamos un objeto llamado Transacción que es el que maneja las operaciones que se realizan en la base de datos, esto se hace usando JDBC/ODBC que es un estándar adoptado por Java para la comunicación de programas con un sistema de base de datos.

Algo que no adoptamos para el desarrollo de este proyecto y que pudo ser añadido como una manera de interactuar con la base de datos es el uso de store procedures.

Los store procedures son buenos por que se ejecutan plenamente en el servidor de base de datos y son más fáciles de probar en cuanto al resultado que devuelven, solucionado cualquier incoherencia.

El inconveniente estaría en que el sistema no sería tan independiente del sistema de base de datos escogido. Por ejemplo para que se ejecute este en un sistema que no soporta store procedures, habría que cambiar algunas líneas de código en la clase Transacción.

Otra cuestión que es importante recalcar en lo que respecta al diseño de la base de datos es que, como se podrá ver, los tipos de datos que en muchos casos se ha escogido para representar a ciertos campos son del tipo texto, ya que los datos de tipo texto nos pareció mas fácil de

manejar, esto debido a que Java provee una serie de métodos para conversión de tipos que se los puede hacer fácilmente en cualquier parte del código de programación.

En cuanto a la asignación de las claves principales en las tablas, muchas claves solo con procesos demostrativos fueron declaradas autonuméricas; sin embargo de ejecutarse en un sistema real debería establecerse algún criterio de asignación de claves dependiendo de los criterios asumidos por el administrador del sistema.

### 5.5.3 Diseño de la Aplicación Servidora.

El diseño fue hecho para que este sirva al cliente, facilitando la programación ya que el cliente puede accesar a las funciones o transacciones que requiera como si fuera en forma local.

Tenemos un objeto encargado de recibir todas las peticiones del cliente el mismo que por medio de sus métodos levantará a los demás objetos obtenidos de nuestro análisis y diseño, estos objetos serán ubicados en el mismo servidor en que se encuentrra la aplicación servidora.

### 5.5.4 Diagrama de Estructuras

La estructura dada o pedida para nuestro proyecto es three Tier, que significa que estara repartid o en 3 partes.

En el primer tier esta el cliente que es la interfaz gráfica del usuario (diferente a la otra interface ORB) que corre sobre un browser, en este se puede realizar cálculos que tenga que hacerse localmente en el cliente sin que sea necesario invocar al servidor. El browser lo que hará es descargar del servidor un applet de java en el que está implementada la lógica de programación del cliente.

El cliente es quien hace uso de los recursos del servidor, por medio del browser que muestra como la interfaz al applet, y este applet utiliza un programa (java) que está implementado en un applet, para invocar a los objetos en el servidor.

En el segundo tier están nuestros objetos, aquí es donde se maneja toda la lógica de la aplicación que se ejecuta en el servidor. Aquí es donde se estarán ejecutando los objetos que hemos obtenido de nuestro análisis y diseño.

Aparte de los objetos obtenidos en nuestro análisis y diseño, también hay otros objetos que son propios de Corba

La implementación de la interface Corba Facturacion es la encargada de receptar todas las necesidades o peticiones.

Y para finalizar en el tercer tier se encuentra nuestro sistema de almacenamiento de datos que en nuestro caso es una base de datos SQL Server; estos sistemas de almacenamiento de datos generalmente tienen implementado toda una cantidad de servicios como son rpc, servicio de transacciones, etc., que son ejecutados internamente y que son transparentes para el usuario.

SQL nos ayuda en la manipulación de datos ya sea para guardarlos, cambiarlos , borrarlos o simplemente consultarlos.
Al hacer los requerimientos al Objeto Transacccion, permite al sistema cambiar de base datos facilmente, pero disminuye en velocidad. En SQL se puede hacer requerimientos directamente en la base de datos, los cuales se llaman Store Procedure que limitan el cambio de base de datos pero mejora en velocidad de accesamiento a los datos.

## 5.6 Implementación del servidor

Para implementar el servidor debemos desarrollar los objetos Corba. Estos objetos serán los encargados de interactuar con nuestra aplicación. Existen dos objetos Corba que cumplirán un papel importante.

El primer paso para el desarrollo de los objetos Corba es la descripción de su IDL.

Los dos objetos Corba que mencionamos serán escritos como interfaces en el archivo IDL.

### 5.6.1 IDL del Proyecto

En el IDL es donde van descritos los módulos y las interfaces con todos los métodos que el cliente va a manipular. La sintaxis del IDL es parecida a la del lenguaje c++.

El IDL se lo escribe en un editor de texto cualquiera, y una vez escrito pasará a un proceso de compilación mediante un compilador IDL-to-Java; en nuestro caso estamos usando el compilador desarrollado por Visibroker de Borlan.

Es importante destacar que un IDL puede ser mapeado a cualquier lenguaje ya sea Java, c, smaltalk o cualquiera de los lenguajes para los que exista un compilador IDL a ese lenguaje, sin que se realice ningún cambio en el código del IDL.

Java no soporta las estructuras, pero la estructura es parte de la sintaxis y la semántica del IDL. Cuando se mapean estructuras al lenguaje java estas

se mapearán como clases, y se las usará en Java sin ningún problema. Las estructuras nos permiten especificar una serie de campos relacionados que pueden ser referenciados con un solo nombre.

Al final de la declaración de cada estructura viene declarada una secuencia, y tiene el siguiente aspecto:

typedef sequence<'nombre de la estructura'> 'nombre de la variable sequence';

El "typedef sequence" es la notación para declarar una secuencia.

Una secuencia no es mas que una variable con un arreglo unidimensional de elementos, un elemento puede ser de cualquier tipo definido en el IDL. Una secuencia puede ser definida a un tamaño máximo o no, por lo que son muy útiles cuando no sepamos cuál es la longitud total de los datos que se van a pasar a un cliente.

El IDL escrito para nuestro proyecto es el siguiente:

```
//System.idl
module Systemm
 {exception SystemmException
 {string r;
  };

 struct clientStruct
 { string fecha_factura;
   string nombre;
   string apellido;
   string cedula;
```

```
  string area;
  string direccion;
  string ciudad;
  string categoria;
  string encontro;
  };
 typedef sequence<clientStruct>  clientseq;

struct areaStruct
{ string codigoarea;
  string numero;
};
typedef sequence<areaStruct> areaseq;

 struct facturaStruct
 { string valor;
  string pendientea;
  string pagado;
  string pendientec;
  string locales;
  string regionales;
  string nacionales;
  string internacionales;
  string celulares;
  string pbasica;
  string iva;
  string deporte;
  string ecapag;
  string tipo_pago;
  string banco;
  string numeroche;
};
typedef sequence<facturaStruct> factseq;

struct pagoStruct
{
 long valor;
 long pendiente_s;
 string tipopa;
 string banco;
 string numeroche;
 };
struct detalle_llamadasStruct
{
 string fecha_llamada;
```

```
 long destinop;
 string destinoz;
 string telefono;
 long minutos;
 string hora;
 long tipo;
 long val;
 };
 typedef sequence<detalle_llamadasStruct> detlseq;
 typedef string are;
 typedef long numcons;
 typedef long result;
 typedef long numconsulta;
 typedef long variable;
 typedef string fecha;
 typedef sequence<string> numeros;

 struct claveStruct
 {
  string telefono;
  string cedula;
 };
 typedef sequence<claveStruct> claveseq;


 interface Facturacion
 {
  clientStruct getdatoscliente(in claveStruct c1,in numcons d1);
  facturaStruct getresumen(in claveStruct c1,in are er,in fecha is);
  result registrar_pago(in pagoStruct d,in are er);
//linea 83
  detlseq getllamadas(in claveStruct c1,in fecha is);
  result enviardetalle(in claveStruct c1,in detalle_llamadasStruct a1,in are er);
  result guardarcliente(in clientStruct en,in areaStruct es);
  result eliminar(in areaStruct es);
  areaseq obtenernumeros(in areaStruct es);
  result refresh(in claveStruct c1,in are er,in numcons a1);
  };

 interface FacturacionService
 { Facturacion resfacturacionObjeto() raises (SystemmException);
   void liberarFacturacionObjeto(in Facturacion facturacionObjeto) raises
(SystemmException);
  };
 };
```

El archivo IDL se compone de modulos, en nuestro proyecto el nombre del modulo es Systemm y está compuesto de una serie de estructuras y de dos interfaces.

### 5.6.2 Diseño de las Interfaces Corba

El diseño consiste de dos interfaces Corba, implementadas en un modulo. Estas dos interfaces al compilarse son objetos Corba en tiempo de ejecución.

A continuación se pasa a describir la implementación de estas dos interfaces.

#### 5.6.2.1 Interface Facturacion

**Interface Facturacion:** Esta interface es la interface principal, con la que el cliente interactúa. Dentro de esta interface están los métodos que un cliente ve (el código java en el applet es el que hace la invocación de los objetos Corba). Los métodos de esta interface a su vez invocarán a los objetos que hemos obtenido y desarrollado de nuestro análisis y diseño previo, que no fueron escritos en IDL sino que fueron escritos en código Java puro para que los manipule esta interface Corba.

Una vez mapeada la interrface Facturacion a código Java y ejecutada esta se comportará como un objeto Corba que levantará a los objetos obtenidos en nuestro análisis y diseño previo.

Métodos:

- **getdatoscliente:** Este método obtiene los datos del cliente cuyo resultado irán registrados en la estructura clientStruct.

  Parámetros de entrada:

  c1,(tipo claveStruct): Aquí van los datos del cliente en base a los cuales se va a realizar la consulta.

  d1,(tipo long): En este parametro va el tipo de consulta que se va a realizar

  Devuelve una estructura de tipo clientStruct en donde va el resultado de la búsqueda.

- **getresumen:** Con este método se obtienen los datos de una consulta de facturación en una fecha determinada.

  Parámetros de entrada:

  c1,(tipo claveStruct): Aquí van los datos del cliente en base a los cuales se va a realizar la consulta.

  er,(tipo string): La categoría a la que pertenece el cliente.

  is, (tipo string): Aquí se manda la fecha en base a la cual se realiza la consulta.

Devuelve una estructura de tipo facturaStruct en donde va el resultado de la búsqueda.

- **registrar_pago:** Con este método se registrará un pago en el sistema.

  Parámetros de entrada:

  d, (tipo pagoStruct): Aqui va la información necesaria para registrar un pago.

  er,(tipo string): Aqui va el numero de teléfono del cliente.

  Devuelve un valor de tipo long en donde se indica si se tuvo exito o no en la operación.

- **getllamadas:** Este método obtiene el detalle de llamadas en una fecha cualquiera.

  Parámetros de entrada:

  c1,(tipo claveStruct): Aqui van los datos del cliente.

  is,(tipo string): Aqui va la fecha en base a la cual se realiza la consulta.

  Devuelve una secuencia basada en una estructura en la que van los detalles de las llamadas.

- **enviardetalle:** Este método va guardar un detalle de llamadas en el sistema.

  Parámetros de entrada:

  c1,(tipo claveStruct): Aqui van los datos del cliente.

al,(tipo detalle_llamadasStruct): Aqui van por medio de la estructura el detalle de la llamada realizada.

er,(tipo string): La categoría a la que pertenece el cliente.

Devuelve un valor de tipo long en donde se indica si se tuvo exito o no en la operación.

- **guardarcliente:** Este método se encarga de registrar un nuevo cliente en el sistema.

  Parámetros de entrada:

  en,(tipo clientStruct): Aqui va la información del cliente a guardar.

  es,(tipo areaStruct): Aqui van los datos respecto al codigo de area y el numero de teléfono.

  Devuelve un valor de tipo long en donde se indica si se tuvo éxito o no con la operación.

- **eliminar:** Este método eliminará a un cliente del sistema.

  Parámetros de entrada:

  es, (tipo areaStruct): Aquí se mandará el número de teléfono y el código de área al que pertenece el cliente.

  Devuelve un valor de tipo long en donde se indica si se tuvo éxito o no con la operación.

- **obtenernumeros:** Este método obtiene los números de teléfonos disponibles en el sistema.

  Parámetros de entrada:

  es,(tipo areaStruct): En esta ocación se usará para mandar el codigo de area en base al cual se obtendrá la información.

  Devuelve una secuencia de tipo areaseq en donde va el resultado obtenido.

- **refresh:** Este método está encargado de actualizar los datos para un nuevo mes de facturación.

  Parámetros de entrada:

  c1,(tipo claveStruct): Aqui van los datos del cliente en base a los cuales se va a realizar la consulta.

  er,(tipo string): La categoría a la que pertenece el cliente.

  a1,(tipo long): un parámetro de estado.

  Devuelve un valor de tipo long que indica si la operación fue llevada a cabo con éxito o no.

#### 5.6.2.2 Interface FacturacionService

Una vez mapeada de IDL a java y luego ejecutada esta interface será un objeto Corba que levantará al objeto Facturacion, con tantas instancias según como

nosotros le especifiquemos. Estas instancias serán enviadas como un parametro variable cuando se levante o ejecute el objeto servidor.

Este objeto trabajará como un TPMonitor generando varios hilos en el servidor para atender a tantos clientes como sea posible.

Métodos:

- **resfacturacionObjeto:** Este método reserva un objeto Facturacion en el servidor para atender a un cliente.
- **liberarFacturacionObjeto:** Este método libera a un objeto Facturacion, despues de que un cliente a terminado de invocarlo.

Una vez escrito el archivo IDL pasamos a su compilación. La compilación consiste en el mapeo del IDL a un lenguaje de programación específico, es decir todo lo que se ha escrito en IDL, en una notación especifica del IDL se convertirá en código que entienda el lenguaje de programación en el que estarán implementados nuestros objetos servidores, para nuestro caso en el lenguaje Java.

En nuestro caso la compilación en el prompt del DOS será como sigue:

prompt> idl2java Systemm.idl

Con la compilación se genera una carpeta con el nombre de nuestro modulo, ya que el modulo generado será un paquete de Java.
Todas estas clases proveen el modelo Corba y el programador no necesariamente tiene que saber como se encuentran internamente implementadas.

El archivo IDL no implementa nada, sino que especifica la estructura general de los objetos servidores mediante sus interfaces. El código de la implementación de la interface se lo escribe en el lenguaje de programación escogido, en nuestro caso el lenguaje de programación escogido es Java.

En el proyecto, el código de programación para la parte servidora que permitirá la ejecución de los objetos Corba se muestra a continuación.

#### 5.6.2.3 FacturacionServiceImpl.java:

Código de programación para la interface FacturacionService escrita en el archivo IDL. Cuando se compile y se ejecute, este será un objeto encargado de poner en funcionamiento al objeto Facturacion con tantas instancias como haya recibido de parámetro.

Es el proveedor de los objetos Facturacion. Reserva o libera objetos de acuerdo al estatus que tenga, siendo el estatus de este : Si en uso o No en uso.

Métodos:

- **Reservacion ObjetoFacturacion:** Es para indicar si el objeto facturacion esta en uso.
- **Liberacion ObjetoFacturacion:** Es para indicar si el objeto facturacion No esta en uso.

**//FacturacionServiceImpl.java**

```
//Comienzo de escritura del archivo FacturacionServiceImpl.java

public class FacturacionServiceImpl extends Systemm._FacturacionServiceImplBase
{ private int maxObjects = 12;
  private int numObjects = 0;
  private FacturacionStatus[] fac = new FacturacionStatus[maxObjects];

  public FacturacionServiceImpl(java.lang.String[] args,
                        java.lang.String name, int num)
  {
  super(name);
  try
  { //Se obtiene una referencia al orb
  org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
  //Precomenzar n Objetos
  numObjects = num;
  for(int i=0;i<numObjects;i++)
  {
   fac[i]=new FacturacionStatus();
   fac[i].ref = new FacturacionImpl("facturacion" + (i+1));
   orb.connect(fac[i].ref);
   }
  }catch(Exception e)
  { System.err.println(e);
    }
  }

  public Systemm.Facturacion resfacturacionObjeto() throws
Systemm.SystemmException
  {
```

```
    for (int i=0;i<numObjects;i++)
    { if (!fac[i].inUse)
      {fac[i].inUse = true;
       System.out.println("facturacion" + (i+1) + "reservado");
       return fac[i].ref;//32
       }
      }
       return null;
   }

  public void liberarFacturacionObjeto(Systemm.Facturacion facturacionObjeto)
throws Systemm.SystemmException
   {  for(int i=0;i<numObjects;i++)
     { if (fac[i].ref == facturacionObjeto)
       { fac[i].inUse = false;
        System.out.println("facturacion" + (i+1) + "liberado");
        return;
        }
     }
System.out.println("Objeto reservado no ha sido encontrado");
        return;
}
}

// Fin de escritura del archivo FacturacionServiceImpl.java
```

#### 5.6.2.4 FacturacionImpl.java:

Código de programación correspondiente a la interface Facturacion escrita en el archivo IDL.

Es el objeto encargado de receptar los diferentes tipos de transacciones por medio del puente ORB, se lo conoce tambien como el objeto coordinador. Es el encargado de llamar a los objetos obtenidos de nuestro analisis y diseño, por medio de todos los métodos que tiene implementado.

**//FacturacionImpl.java**

```
// Comienzo de escritura del archivo FacturacionImpl.java
import java.util.*;
public class FacturacionImpl extends Systemm._FacturacionImplBase
{
 Factura_r fac;
 private String instanceName;
 private int tr;
 public Transaccion x;
 public FacturacionImpl(java.lang.String name)
 {
     super(name);
     try
     {
     //Se obtiene el codigo del mes que está registrado en la
     //base de datos como mes de facturación
     x = new Transaccion();
     x.open();
     fac = new Factura_r(x);
     tr = fac.obtener_mesfacturacion();
     instanceName = name;
     } catch(Exception e)
     { System.out.println("System Exception");
     }
 }

 public FacturacionImpl()
 {
     super();
          try
     {
     //Se obtiene el codigo del mes que está registrado en la
     //base de datos como mes de facturación
     x = new Transaccion();
     x.open();
     fac = new Factura_r(x);
     tr = fac.obtener_mesfacturacion();
     //instanceName = name;
     } catch(Exception e)
     { System.out.println("System Exception");
     }

 }
```

```
public Systemm.clientStruct getdatoscliente(Systemm.claveStruct c1,int d1)
{  Cliente s;//32
 Systemm.clientStruct datos;
 try{ datos = new Systemm.clientStruct();
    s= new Cliente(x);
    if(d1 == 1)
     datos = s.encontrar(c1.telefono,c1.cedula);
     if (d1 == 2)
     datos = s.encontrar2(c1.telefono);
     if (d1 > 2)
      return null;
     return(datos);
 } catch (Exception e)
 { System.out.println("Algún error ha ocurrido FacturacionImpl");
 return null;
 }
 }

 public Systemm.facturaStruct getresumen(Systemm.claveStruct c1,java.lang.String
er,java.lang.String is)
 {  int nt;
   Consumo z;
 // private Trasaccion m;
   Systemm.facturaStruct datoss;
 // m = new Transaccion();
    try
    {
 nt = x.obtener_codigofecha(is);
 if (nt == -1)
 return null;


    //datos = new Systemm.facturaStruct();
    //Se sobreentiende que ya se ha iniciado detalle factura
    z = new Consumo(nt,c1.telefono,er,x);
    datoss = z.obtener_detalleconsumo(is);
    return datoss;
    } catch (Exception e)
     { System.out.println("Algún error ha ocurrido en FacturacionImpl");
     return null;
     }
    }

 public Systemm.detalle_llamadasStruct[] getllamadas(Systemm.claveStruct
c1,java.lang.String is)
```

```
{ // private Transaccion z;
  Systemm.detalle_llamadasStruct[] datos;
  try{
   Vector datosx = new Vector();
   //z = new Transaccion();
   datosx = x.obtener_detallellamadas(c1.telefono,is);
   datos = new Systemm.detalle_llamadasStruct[datosx.size()];
   datosx.copyInto(datos);
   return(datos);
  } catch (Exception e)
  {System.out.println("Algún error ha ocurrido en FacturacionImpl");
  return null;
  }
  }

  public int enviardetalle(Systemm.claveStruct c1,Systemm.detalle_llamadasStruct
a1,java.lang.String er)
  {
   int b,cr;
   Consumo_local r;
   Consumo_internacional xx;
  try
  {
   if (a1.tipo == 4)
    { xx=new Consumo_internacional(tr+1,c1.telefono,er,x);
     b=xx.iniciar_datos();
      if (b== -1)
       return -1;

b=xx.guardar(a1.fecha_llamada,a1.minutos,a1.telefono,a1.hora,a1.destinoz,a1.destino
p);
      if (b==-1)
      { System.out.println("Algún error ha ocurrido en FacturacionImpl");
       return -1;
       }
    }
    else
    { r= new Consumo_local(tr + 1,c1.telefono,er,x);
     System.out.println("se creo el constructor de consumo local\n");
     System.out.println("valor de tr " + tr + "valor de telefono  " + c1.telefono +
"valor categoria " + er + "\n");

     b=r.iniciar_datos();
    System.out.println("Se iniciar¾n los datos\nn");
     if (b== -1)
```

```
        return -1;

b=r.guardar(a1.fecha_llamada,a1.minutos,a1.telefono,a1.hora,a1.tipo,a1.destinoz);

      }//112
      return b;
    } catch(Exception e)
      { System.out.println("Algún error ha ocurrido en FacturacionImpl");
        return -1;
      }
      }

 public int guardarcliente(Systemm.clientStruct en,Systemm.areaStruct es)
 { //private Transaccion x;
   int b;
  try{
   //x=new Transaccion();
   b=x.almacenar_cliente(en,es);
   if (b==-1)
   {  System.out.println("Algún error ha ocurrido en FacturacionImpl almacenar");
     return -1;
   }
   b=x.eliminar_numero(es);
   if (b==-1)
   {  System.out.println("Algún error ha ocurrido en FacturacionImpl eliminar");
     return -1;
   }
   return b;
   } catch(Exception e)
   { System.out.println("Algún error ha ocurrido en FacturacionImpl");
    return -1;
     }
   }

public int eliminar(Systemm.areaStruct es)
{  Cliente t;
   int b;
  try
  { t= new Cliente(x);
    b= t.eliminar(es.numero);
    if (b==-1)
    { System.out.println("Alg·n error ha ocurrido en FacturacionImpl");
     return -1;
     }
    b=t.disponer(es.numero,es.codigoarea);
```

```
 if (b==-1)
  { System.out.println("Algún error ha ocurrido en FacturacionImpl");
    return -1;
    }
 return b;
 } catch(Exception e)
 { System.out.println("Algún error ha ocurrido en FacturacionImpl");
  return -1;
  }
 }

public Systemm.areaStruct[] obtenernumeros(Systemm.areaStruct er)
{ //String[] datos;
 //private Transaccion t;
  int b;
 try {     Systemm.areaStruct[] datos;
           Vector datosx= new Vector();
      //t=new Transaccion();
      datosx = x.obtener_numerosdispon(er.codigoarea);
      //datos = new String[datosx.size()];
      datos = new Systemm.areaStruct[datosx.size()];
      datosx.copyInto(datos);
      return datos;
      } catch(Exception e)
      { System.out.println("Algún error ha ocurrido en FacturacionImpl");
       return null;
        }
 }

public int refresh(Systemm.claveStruct c1,java.lang.String er,int inser)
{  Factura_r f;
  int b;
 try
  { f=new Factura_r(x);
   b=f.iniciar_detallefactura(tr,c1.telefono,er,inser);
   if (b==-1)
   { System.out.println("Algún error ha ocurrido en FacturacionImpl");
    return -1;
    }
  } catch(Exception e)
  { System.out.println("Algún error ha ocurrido en FacturacionImpl");
   return -1;
   }
   return b;
  }
```

```
public int registrar_pago(Systemm.pagoStruct d,java.lang.String er)
{ //private Transaccion x;
  int b;
 try
 { //x=new Transaccion();
  b=x.guardar_pago(d,tr,er);
  if (b==-1)
  { System.out.println("Algún error ha ocurrido en FacturacionImpl");
  return -1;
  }
  return b;
 } catch(Exception e)
  { System.out.println("Algún error ha ocurrido en FacturacionImpl");
   return -1;
   }
 }
}
```

// Fin de escritura del archivo FacturacionImpl.java

#### 5.6.2.5 SystemmServer.java:

Este archivo no corresponde directamente, con el código escrito en el archivo IDL. En este archivo va el código que invocará al objeto FacturacionService (Facturacion Service fue escrito en el IDL), y lo levantará, poniéndolo a ejecutar como proceso independiente.

Cuando el SystemmServer (no escrito en el IDL) se ejecute, recibirá como parámetro un valor que describe el número de instancias al objeto Facturacion (escrito en el IDL); este parámetro será enviado por el administrador del servidor. Luego el SystemmServer pasa este parámetro al objeto FacturacionService (escrito en el IDL) el mismo que hace las funciones de un

TP Monitor, poniendo en una serie de hilos a ejecutar al objeto Facturacion (escrito en el IDL) con tantas instancias como el administrador del servidor haya mandado como parámetro al objeto SystemmServer.

A continuación se demuestra el código de programación de cada uno de los objetos descritos anteriormente.

**//SystemmServer.java**

```
//Comienzo de escritura del archivo SystemmServer
import org.omg.CosNaming.*;

class SystemmServer
{
 static public void main(String[] args)
 {
  int instancias;
  try
  { if(args.length == 0)
       instancias = 5;
    else
       instancias = Integer.parseInt(args[0]);
       //inicializar el orb
       org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
       //se crea el objeto FacturacionService
       FacturacionServiceImpl proveedor = new
FacturacionServiceImpl(args,"service",instancias);
       //se exporta el objeto creado
       orb.connect(proveedor);
       //se obtiene una referencia al servicio de nombramiento
       org.omg.CORBA.Object nameServiceObj =
orb.resolve_initial_references("NameService");
       if (nameServiceObj == null)
       {  System.out.println("nameServiceObj = null");
          return;
       }
       org.omg.CosNaming.NamingContext nameService =
org.omg.CosNaming.NamingContextHelper.narrow(nameServiceObj);
       if (nameService == null)
```

```
      { System.out.println("nameService = null");
        return;
    }
    //Adhiero el objeto al servicio de nombramiento
    NameComponent[] SystemServerName = {new
NameComponent("facturacion","")};
    nameService.rebind(SystemServerName,proveedor);
    //Espera en hilo hasta terminar
    Thread.currentThread().join();
  } catch(Exception e)
  { System.err.println(e);
  }
 }
}

// Fin de escritura del archivo SystemmServer.java
```

#### 5.6.2.6 Herencia de clases.

FacturacionServiceImpl y FacturacioImpl heredan de las clases obtenidas en el proceso del mapeado del IDL a java. Esta herencia se manifiesta por la palabra extend escrita en la declaración de la clase.

FacturacionImpl hereda de la clase Systemm._FacturacionImplBase (Systemm es el nombre del modulo declarado en el IDL y que se mapeó como un paquete de Java). Para explicarlo de manera fácil la clase _FacturacionImplBase contiene el código necesario que permite la comunicación del lado del servidor (del objeto servidor Facturacion) con el cliente que haga la invocación del objeto servidor Corba. Es decir esta clase

hace la transparencia al programador de la comunicación con el cliente. Esta clase fue generada automáticamente en el proceso de mapeado del IDL a java.

Para FacturacionServiceImpl es lo mismo. Este en cambio hereda de la clase Systemm._FacturacionServiceImplBase, otra clase obtenida en el proceso de mapeado de IDL a Java. Para cada interface escrita en el IDL se generan una serie de clases entre las que están las clases terminadas con la palabra ImplBase.

El objeto Facturacion invocará a todo los objetos que se obtuvieron del análisis y diseño de objetos que se hizo en la primera etapa.

El código de los objetos obtenidos del análisis y diseño de objetos ya se describió en la primera etapa (Diseño y descripción de clases), esta parte de la explicación solo corresponde a los objetos que se ejecutan como procesos servidores.

Lo que viene luego es la compilación de todos estos archivos para poderlos ejecutar.

Cuando se ejecute el SystemmServer a este se le pasará el parámetro que le indicará el número de instancias que debe generar del objeto Facturacion, es decir este valor indicará a cuantos clientes podrá atender el servidor.

Este parámetro podrá ser dado como indica el siguiente ejemplo:

prompt> start java SystemmServer 6

Donde el valor de 6 indica el número de instancias que el servidor generará del objeto Corba Facturacion para atender al cliente.

## 5.7 Diseño del Servidor de Base de Datos

### Descripción de las tablas

Los nombres de las tablas y los campos están escritos exactamente como se los ha nombrado en la base de datos, en algunos casos pueden aparentar falta ortográfica, ya que se ha preferido mantener su nombre original.

**Tabla cliente:**

Descripción de campos:

-**codigo_cliente:** de tipo autonumérico, contiene una clave única asignada en secuencia por el sistema a cada usuario.

-**nombre**: de tipo texto, contiene el nombre del cliente.

-**apellido:** de tipo texto, contiene el apellido del cliente.

-**cedula:** de tipo texto, contiene el numero de cédula del cliente.

-**direccion**: de tipo texto, contiene la dirección de la ubicación donde reside el cliente.

-**ciudad:** de tipo texto, contiene la ciudad en la que reside el cliente.

**Tabla categoria**:

Descripción de campos:

**-idcategoria:** de tipo autonumérico, contiene una clave autonumérica asignada a cada categoría.

**-descripcion:** de tipo texo, contiene una breve descripción de la categoría y a que tipo de clientes de refiere.

**-tipo:** de tipo texto, se refiere a la clasificación de los diferentes tipos de categorías que existen.

**Tabla numeros:**

Descripción de campos:

- **idnumero:** de tipo autonumérico, contiene una clave única asignada a un número de teléfono

- **telefono:** de tipo texto, contiene el número de teléfono del cliente.

- **area:** de tipo texto, contiene el codigo de area en el que está registrado ese número de teléfono.

- **codigo_cliente:** de tipo numérico, contiene la clave foranea que hace referencia a la tabla cliente.

- **idcategoria:** de tipo numérico, contiene la clave foranea que hace referencia a la tabla categoria.

**Tabla destinos:**

Descripción de los campos:

- **iddestinos:** de tipo numérico, asigna la clave a los destinos internacionales ubicados en esta tabla.

- **Grupo:** de tipo texto, los destinos internacionales se encuentran agrupados y su tarifa internacional se basa en los destinos ingresados en esta tabla.

**Tabla tarifas:**

Descripción de los campos:

- **tarifas:** de tipo autonumérico, esta es la clave de la tabla.

- **valor minutos:** de tipo numérico, contiene el valor por minuto de una llamada según la categoria y el tipo de llamada y el destino en caso de ser una llamada de tipo internacional.

- **tarifabasica:** de tipo numérico, contiene la tarifa basica de una llamada según la categoria.

- **minutosgratis:** de tipo numérico, contiene los minutos gratis asignados a una categoria de terminada.

- **idcategoria:** de tipo numérico, es la clave foranea que hace relación con la tabla

categoria.

- **idfecha:** de tipo numerico, es la clave foranea que hace relación con la tabla fecha_facturacion. Debido a que las tarifas pueden cambiar con el tiempo y

con el objetivo de almacenar información de tarifas a través del tiempo se hace relación con esta tabla.

- **idtipo:** de tipo numerico, es la clave foranea que hace relación con la tabla tipollamadas.
- **iddestino:** de tipo numerico, es la clave foranea que hace relación con la tabla destinos.

**Tabla tipollamadas:**

Descripción de los campos:

- **idtipo:** de tipo numérico, es la clave de la tabla tipollamadas.
- **tipotarifa:** de tipo texto, contiene la descripción del tipo de llamada.

**Tabla recargos:**

Descripción de los campos:

- **idrecargo:** de tipo autonumérico, es la clave de la tabla recargos.
- **iva:** de tipo numérico,contiene un valor asignado a este rubro.
- **deporte:** de tipo numérico, contiene un valor asignado a este rubro.
- **ecapag:** de tipo numérico, contiene un valor asignado a este rubro.
- **idfecha:** de tipo numérico, clave foranea que hace referencia a la tabla fecha_facturacion con el objetivo de almacenar datos que puedan ser consultados con el pasar del tiempo.

**Tabla detalle_llamadas:**

Descripción de los campos:

- **codigo_detalle:** de tipo autonumérico, es la clave de la tabla detalle_llamadas.

- **tipollamadas:** de tipo numérico,es la clave foranea que hace relación con la tabla tipollamadas.

- **fecha_llamadas:** de tipo texto, contiene la fecha en que realizó una llamada un cliente dado.

- **destino:** de tipo texto, contiene el lugar de destino hacia el que realizó una llamada un cliente determinado.

- **telefono:** de tipo texto, contiene el numero de teléfono hacia el que realizó una llamada un cliente determinado.

- **minutos:** de tipo numérico, contiene el numero de minutos de la llamda que realizado un cliente determinado.

- **hora:** de tipo texto, contiene el tiempo formateado en el que se realizó una llamada.

- **valor:** de tipo numérico, contiene el valor de llamada realizada en el tiempo específico en que duró esta.

- **codigonumero:** de tipo numérico, es la clave foranea que hace referencia a la tabla numero.

- **codigofecha:** de tipo numérico, es la clave foranea que hace referencia a la tabla fechafacturacion, con el propósito de registrar las llamadas realizadas en un tiempo de terminado.

**Tabla detallefactura:**

Descripción de los campos:

- **codigo_factura:** de tipo autonumérico, es la clave principal de la tabla detallefactura.
- **valorfactura:** de tipo numérico, campo que contiene el valor total de la factura en la fecha especificada por la relación.
- **pendiente_sig:** de tipo numérico, valor de la factura que queda pendiente de pagar para el mes siguiente.
- **valorpagado:** de tipo numérico, contiene el valor pagado por el cliente en la fecha determinada.
- **pendiente_ant:** de tipo numérico, contiene el valor que había quedado pendiente de pagar en el mes anterior.
- **llamadaslocales:** de tipo numérico, contiene el valor total de las llamadas locales realizadas en la fecha indicada por la relación.
- **llamadasregionales**: de tipo numérico, contiene el valor total de las llamadas regionales realizadas en la fecha indicada por la relación.
-**llamadasnacionales:** de tipo numérico, contiene el valor total de las llamadas nacionales realizadas en la fecha indicada por la relación.

- **llamadasinternacionales:** de tipo numérico, contiene el valor total de las llamadas internacionales realizadas en la fecha indicada por la relación.

- **llamadascelulares:** de tipo numérico, contiene el valor total de las llamadas a celulares realizadas en la fecha indicada por la relación.

- **idnumero:** de tipo numérico, es la clave foranea que hace referencia a la tabla numero.

- **idfecha:** de tipo numérico, es la clave foranea que hace referencia a la tabla fecha_facturacion.

- **tipo_pago:** de tipo texto, contiene un indicativo del tipo de pago en que se ha llevado a cabo la transacción.

- **banco:** de tipo texto, contiene el nombre del banco al que pertenece un cheque.

- **numeroche:** de tipo texto,contiene el número de cheque en el que fue realizado el pago.

**Tabla numeros_disponibles:**

Descripción de los campos:

- **idnumero:** de tipo autonumérico, es la clave de la tabla.

- **telefono:** de tipo texto, contiene el número de teléfono disponible.

- **area:** de tipo texto, contiene el codigo de area al que pertence ese número de teléfono.

### 5.7.2 Relacion de la Base de datos.

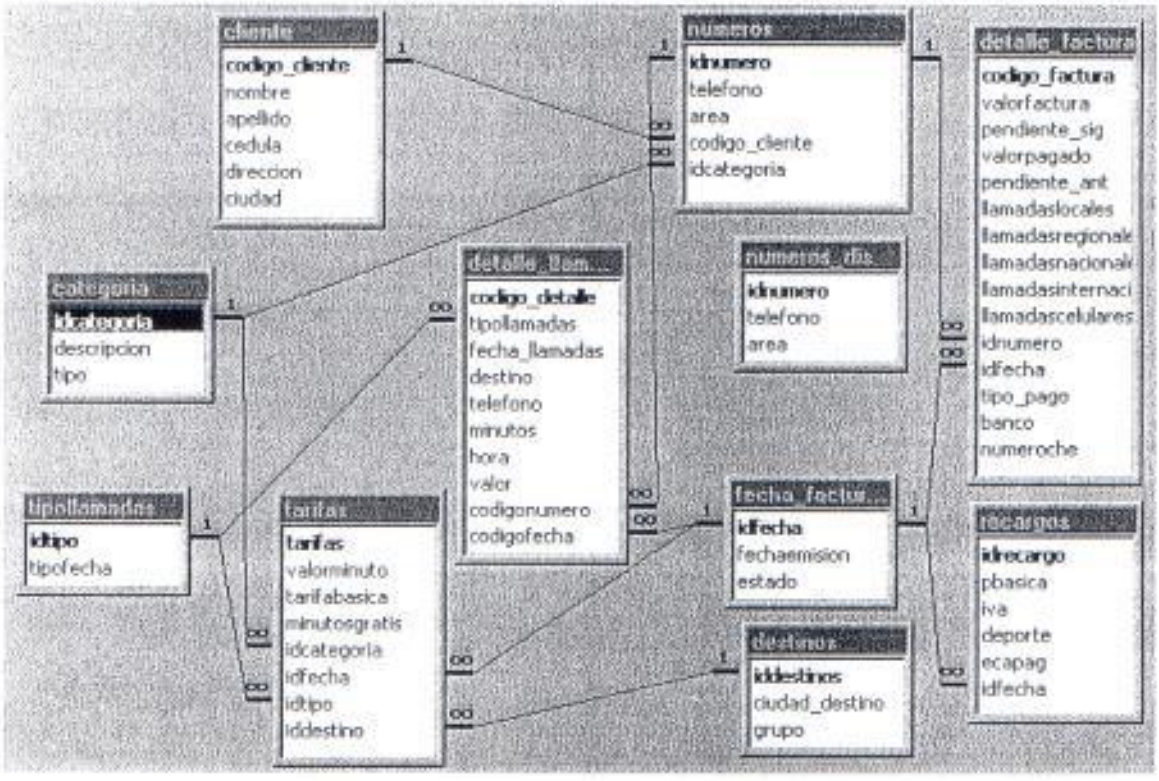



Figura : Relaciones de Tablas en Base de Datos

### 5.5.4.1 Descripcion de Relaciones.

**CATEGORIANUMEROS**

| **categoria** | | | **numeros** |
|---|---|---|---|
| idcategoria | 1 | □ | idcategoria |

Atributos: Uno a varios
Atributos: Exigir, Actualizaciones en cascada

## CATEGORIATARIFAS

| categoria | | | tarifas |
|---|---|---|---|
| idcategoria | 1 | □ | idcategoria |

Atributos: Uno a varios
Atributos: Exigir, Actualizaciones en cascada

## CLIENTENUMEROS

| cliente | | | numeros |
|---|---|---|---|
| codigo_cliente | 1 | □ | codigo_cliente |

Atributos: Uno a varios
Atributos: Exigir, Actualizaciones en cascada, Eliminaciones en cascada

## DESTINOSTARIFAS

| destinos | | | tarifas |
|---|---|---|---|
| iddestinos | 1 | □ | iddestino |

Atributos: Uno a varios
Atributos: Exigir, Actualizaciones en cascada

## FECHA_FACTURACIONDETALLE_FACTURA

| fecha_facturacion | | | detalle_factura |
|---|---|---|---|
| idfecha | 1 | □ | idfecha |

Atributos: Uno a varios
Atributos: Exigir, Actualizaciones en cascada, Eliminaciones en cascada

## FECHA_FACTURACIONDETALLE_LLAMADAS

| fecha_facturacion | | | detalle_llamadas |
|---|---|---|---|
| idfecha | 1 | □ | codigofecha |

Atributos: Exigir, Actualizaciones en cascada, Eliminaciones en cascada

Atributos: Uno a varios

## FECHA_FACTURACIONRECARGOS

| fecha_facturacion | | | recargos |
|---|---|---|---|
| idfecha | 1 | □ | idfecha |

Atributos: Exigir, Actualizaciones en cascada, Eliminaciones en cascada
Atributos: Uno a varios

## FECHA_FACTURACIONTARIFAS

| fecha_facturacion | | | tarifas |
|---|---|---|---|
| idfecha | 1 | □ | idfecha |

Atributos: Uno a varios
Atributos: Exigir, Actualizaciones en cascada, Eliminaciones en cascada

## NUMEROSDETALLE_FACTURA

| numeros | | | detalle_factura |
|---|---|---|---|
| idnumero | 1 | □ | idnumero |

Atributos: Uno a varios
Atributos: Exigir, Actualizaciones en cascada, Eliminaciones en cascada

## TIPOLLAMADASDETALLE_LLAMADAS

| tipollamadas | | | detalle_llamadas |
|---|---|---|---|
| idtipo | 1 | □ | tipollamadas |

Atributos: Uno a varios
Atributos: Exigir, Actualizaciones en cascada

## TIPOLLAMADASTARIFAS

| tipollamadas | | | tarifas |
|---|---|---|---|
| idtipo | 1 | □ | idtipo |

Atributos: Exigir, Actualizaciones en cascada

| | |
|---|---|
| *Atributos:* | Uno a varios |

## 5.8 Diseño del Cliente

Se analizó varias posibilidades de cómo hacer el cliente, ya que podria ser de varios applets sencillos que contengan una transaccion a la vez. Pero se tenía una dificultad con este tipo de cliente. Otra posibilidad fue la que hemos escogido.

Se lo realizó con el enfoque de un Applet que pueda contener varios paneles y de esta manera superar el inconveniente de poder mantener información de una transaccion anterior de un cliente sin tener la necesidad de accesarlo otra vez.

El applet del Cliente fue realizado con la ayudad de Visual Café permitiendo tener un Tab de paneles para todas las opciones de transacción.
El cliente solo con dar un Click del ratón sobre botones en la interfaz grafica llama a funciones que se encuentran en objetos ubicados en el servidor. Por medio de la interface se accesa a estos objetos.
El código de la interacción con los objetos Corba no presenta ninguna dificultad para el programador ya que se trata a los objetos servidores como si fueran objetos locales.

En el Applet se realizaron varios mensaje de Error que sirven para evitar realizar las transacciones en forma incorrecta. Estos mensajes de errores, en su mayoría

no llaman a ningún objeto sino que comprueba su veracidad dentro del propio Applet. Los erores que detecta son de formato , nombre, incompletos, campos nulos, etc.

Los Paneles que contiene este Applet son de Ingresar Consumo, Consultar datos, Recepcion de Pagos, Detalle de Llamadas, Asignacion de Numeros y Baja de Numeros.

### 5.8.1 Applet Principal

Es el nombre que se le ha dado al Applet, encargado de facilitar al Cliente una interfaz gráfica para el manejo de las transacciones en una manera amigable. Es el encargado de llamar a la función que nos conecta con el servidor a travez de su interface ORB.

## 5.8.2 Descripción de la Interfaz gráfica del Cliente.

La Interfaz gráfica consta de 2 partes que se encuentran dentro de un Applet:
La primera es un panel que hemos denominado Principal, ya que es donde se muestra el número de teléfono, y los datos del Cliente, como son Nombre, Apellido, Dirección, Ciudad, Categoría, y Fecha de Facturación.

La segunda es el TAB panel, el mismo que contiene 6 Paneles y consta de las 6 transacciones: Ingreso, Consulta, Recepción de Pagos, Detalle de Consumo, Asignación de Números y Baja de Números.

### 5.8.3 Inicio de la Aplicación.

Para iniciar la aplicación se debe de ingresar el número de teléfono y presionar Aceptar, en el Panel Principal. Si los datos correspondientes a este número existen, entonces se muestran los datos del cliente en el Panel Principal y en las otras 6 opciones, se muestran o guardan los datos correspondientes a este cliente.

Así pasamos a la descripción de las operaciones de cada transacción.
Se requiere que realice 3 operaciones principales y 2 opcionales:

### 5.8..4 Ingreso de Consumo:

Se realizaran 2 tipos de ingresos, (local y de larga distancia) que corresponden al consumo telefónico de un cliente.
Si es local, Nacional, Regional o Celular, el consumo se transforma en sucres de acuerdo al precio de cada tipo.

Si es de Larga Distancia, se indicará el lugar de la región a donde se llama, el consumo se valora de acuerdo al tiempo de la llamada y la localidad con la que se comunicó (el cual se transforma en sucres de acuerdo al tiempo y el precio que se encuentre asignado a esa localidad)

### 5.8..5 Consulta:

Esta transacción debe mostrar en pantalla el detalle del consumo del mes de facturación o del mes escogido (tanto llamadas locales como de larga distancia). Se brindará una información total de cada uno de los tipos de llamada con su valor respectivo y el valor de sus adicionales. Además de se podrá analizar el valor adeudado que viene arrastrando, junto con el valor adeudado que deje para el mes siguiente. Por definición se mostrarán los datos del mes de facturación pero el usuario será libre de cambiar la fecha para consultar los datos de algún otro mes existente en el sistema.

### 5.8..6 Recepción de Pagos:

La transacción muestra el monto adeudado, y permitirá realizar el ingreso del pago. Si el usuario no paga toda la cantidad, el saldo se incrementará a la factura del siguiente mes.

Se podrán realizar dos tipos de pago: en efectivo y en cheque. Si el pago es hecho en cheque se desplegarán las opciones correspondientes a este tipo de pago como son el banco, la fecha del cheque y el numero del cheque.

### 5.8.7 Detalle de llamadas:

Por medio de esta opción el usuario podrá consultar el detalle de llamadas realizadas en un mes determinado. Por definición se desplegarán los datos correspondientes al mes de facturación escogido, pero el usuario podrá consultar detalle de llamadas de un mes distinto al mes de facturación.

### 5.8.8 Asignación de Números:

Por medio de esta opción se realizará la asignación de números a los clientes. Se deberá ingresar primero el código de área al que pertenecerá el número, luego el sistema devolverá una serie de números disponibles para este código de área. De acuerdo a esto de deberá escoger él numero deseado y llenar en el panel Principal los datos del usuario al que desea asignar el número. Finalmente se presiona el botón asignar y los datos serán guardados en el sistema.

### 5.8.9 Baja de números:

Con esta opción se dará de baja al número de teléfono asignado a un cliente. El número pasa a estar disponible para un nuevo abonado. Esta transacción elimina la relación que existe entre el cliente y el número del abonado. No elimina los datos del usuario dado que este puede tener asignado otro número.

**Diagrama de funcionamiento.**

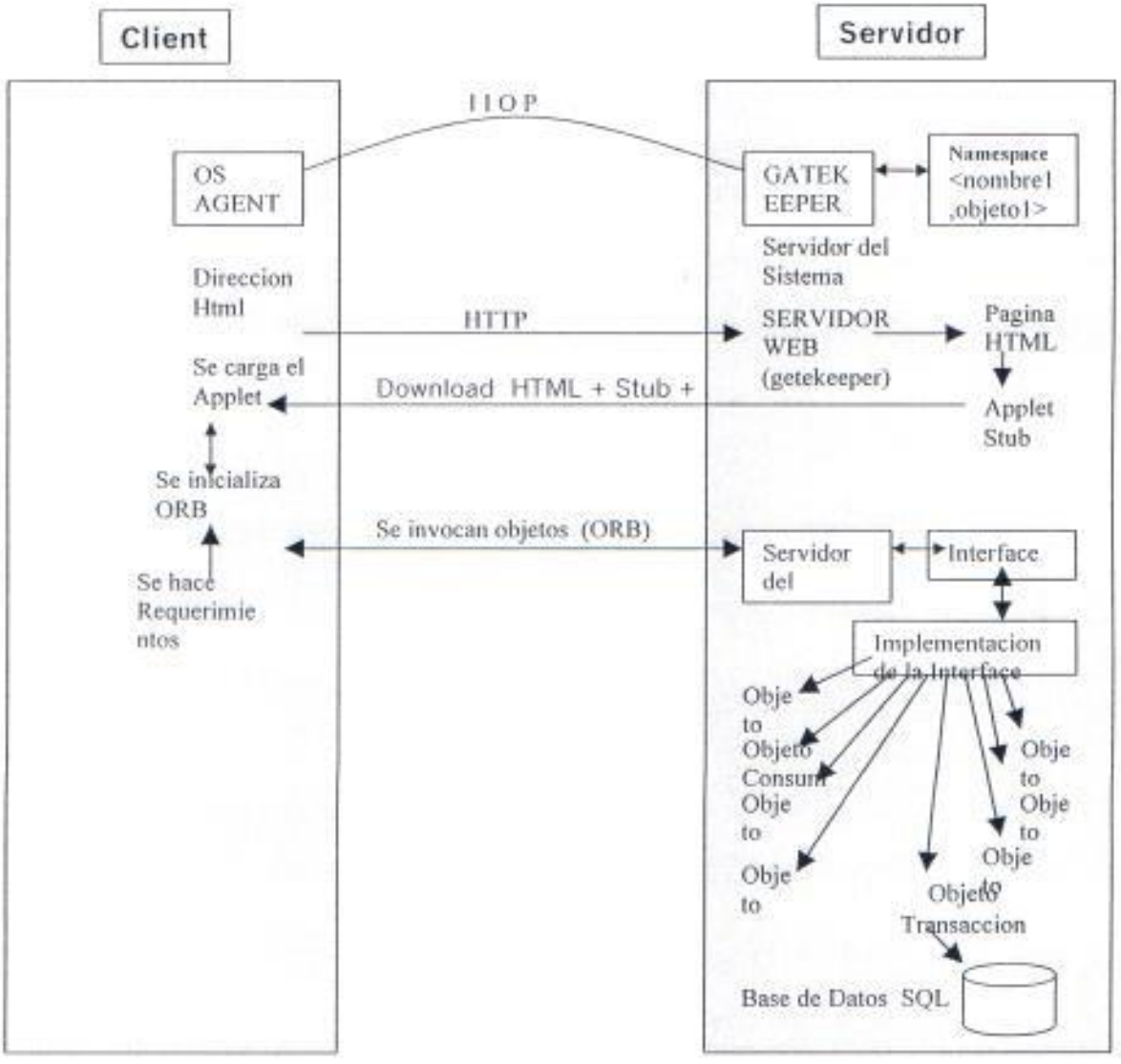



Figura: Funcionamiento de la aplicación Facturacion de Empresa Telefonica.

## 5.9 Explicación del Diagrama de funcionamiento.

### 5.9.1 Instalación y configuración previa

Primero se debe tener bien instalado Visibroker for Java , Visibroker for Naming Service, Visual Café , y Netscape versión 4.5 con parche o un navegador Superior.

Luego se crea en el disco duro un Folder o carpeta donde se guarda:

- La pagina Html que llamara al applet " Principal.html"
- El Applet que correrá en el browser del cliente " Principal.class"
- Las clases obtenidas del análisis y diseño
- La precompilacion del Idl (Stub, skeleton, helper, holder, Impl, example) que son los que nos permiten tener el puente ORB. Cuando se carga el ORB en el Servidor nos referimos a que el servidor muestra los métodos con los que dispone la aplicación, cuando se carga el ORB en el cliente nos referimos a que el cliente busca en el servidor, dentro del servicio de nombramiento los métodos con que dispone la aplicación.

### 5.9.2 Puesta en ejecución

Después de haber realizado la instalación previa , levantamos lo necesario para correr la aplicación:

- Se levanta el Os Agent, que es el encargado de encontrar los objetos servidores Corba.
- Se levanta el Naming Service, que permite registrar el nombre de la interface de la aplicación con el servicio de nombramiento, quedando en el NameSpace relacionado nombre a objeto.
- Se levanta el SystemmServer, este hace instancias de la implementación de la interface FacturacionImpl, que contiene los métodos con los que interactúan con el. cliente
- Se levanta el Gatekeeper, que se encarga de iniciar el IIOP que permite comunicación entre cliente y servidor. El gatekeeper además sirve como servidor Web , que permite al servidor interpretar y ejecutar las paginas Html.
- Finalmente se levanta el Browser, al cual indicamos la dirección del servidor y de la pagina que buscamos.

La pagina Html se encarga de buscar en el servidor el Applet, para mostrarlo en el Browser, pero primero este es bajado (copiado) al cliente, junto con los archivos necesarios como es el Stub, que nos permitira establecer la comunicación ORB.

El Applet es mostrado en el browser, el ORB es cargado en el cliente. La aplicación queda lista para hacer requerimientos.

Se ingresa un número de teléfono, automáticamente la interface gráfica llama por medio de IIOP a la interface Corba, que a su vez llama a la implementación de esta interface "FacturacionImpl" .

La implementación de la interface se encarga de levantar los objetos que necesita para hacer uso de sus métodos.

# CONCLUSIONES Y RECOMENDACIONES.

Debido a que Corba es una herramienta nueva, el tiempo en aprendizaje es un factor importante a invertir.

Debido a que Corba nos presenta un ambiente orientado a objetos, la programación tiene que hacerse de la misma manera. Nosotros hemos tenido mas dificultades en el diseño e implementación de objetos, que en lo referente al diseño de la forma de comunicarse por medio de Corba.

La utilización de objetos Corba es más sencilla en los clientes pero mas compleja en el servidor, pero a su vez al ser mas compleja en el servidor es mas especifica.

Gracias a esta forma de programación las invocaciones en el cliente se las realiza como si se tratara de invocaciones a funciones locales. Siendo transparente los detalles de comunicación para el programador Orb.

El sistema está diseñado de tal manera que puede ser ampliado fácilmente dado a su estructura de orientación a objetos.

La programación con Corba es la mas propicia en lo referente a programación orientada a objetos a través de la red y es muy aplicable dado que utiliza a Java como su forma de comunicación.

La competencia a Corba es DCOM que está respaldado por Microsoft quedando con ventaja este último por su gran soporte de usuarios.

Se recomienda hacer uso de herramientas como visual café para lo que es la interfaz gráfica, sin embargo la codificación de los objetos se hará programando sin ayuda visual.

# BIBLIOGRAFÍA


1. GRIFFITH & CHAN & F.ISAI. 1001 Tips para Programar con Java(1ra edicion en español Mexico 1998; Mc Graw Hill

2. JAMSA K. Ph. D. , Java Now, (Mc Graw Hill, New York)

3. ORFALI R. & HARKEY D., Client / Server Programing with JAVA and CORBA (2da Edicion, New York: John wiley & Sons, Inc, 1998)

4. ORFALI R. & HARKEY D & EDWARDS J. Essencial Client / Server Survival Guide (2da Edicion, New York: John wiley & Sons, Inc, 1998)