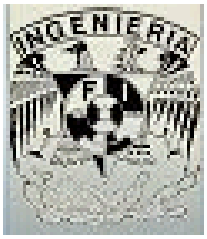
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

“SISTEMA DE COMERCIALIZACIÓN EN LÍNEA UTLIZANDO TECNOLOGÍAS WEB”

T E S I S P R O F E S I O N A L

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN



P R E S E N T A:
RICARDO HUERTA DE LA TEJERA

DIRECTOR DE TESIS: ING. ALEJANDRO MANCILLA ROSALES

CIUDAD UNIVERSITARIA, MÉXICO, D.F. 2004

Dedicatorias

A mi madre:

La parte más importante y querida de mi vida quien con su compañía, constancia y cariño me ha impulsado para alcanzar todos mis sueños y éste muy en especial pues ella sabe el significado de este esfuerzo.

A mi hermano Luis Gerardo:

Él ha sido una parte muy importante y querida en mi vida pues él ha sido mi amigo y mi guía, él me ha enseñado muchas cosas que me han permitido formar mi carácter y enfrentar la vida con el deseo y coraje de triunfar.

A Mariana Silva:

A la mujer que he querido desde hace mucho tiempo, a ella que le dió luz y alegría a mi vida y a la que siempre llevaré en mi corazón aunque ya no esté más conmigo.

A Daniel y Corazón de María:

Mis hernamos muy queridos con los cuales he compartido triunfos y caidas y quienes de alguna manera son partícipes de este esfuerzo pues han seguido de cerca mis pasos y quiero que sean más exitosos de lo que ahora son.

Agradecimientos

A Dios:

Por ser la luz que guía mi camino y permitirme cerrar este ciclo de mi vida.

A mi madre:

Por su dedicación, amor y cuidados que ha tenido para conmigo desde niño, por su ejemplo de constancia y corage para enfrentar la vida.

A mi hermano Luis Gerardo:

Por su compañía, cariño y cuidados en gran parte de mi vida, quien con su ejemplo sentó las bases para formar mi carácter y ser una persona de bien.

A Daniel y Corazón de María:

Por su compañía, cariño y apoyo incondicional en todo momento de mi vida.

A Miguel López Guerrero y Dinóra Najera Diaz:

Por amistad y su ejemplo de dedicación y esfuerzo que me ha impulsado a conseguir mis sueños.

A mis amigos:

Por su amistad, su paciencia y su apoyo incondicional.

A mi tutor:

Ing. Alejandro Mancilla Rosales por su amistad, por el tiempo que me dedicó para lograr este proyecto, por su ejemplo de dedicación y esfuerzo.

Contenido

PARTE I
TEMA DE TESIS

Contenido

PARTE II
APÉNDICES

CAPÍTULO 1

Planteamiento del problema

Actualmente existe una creciente demanda de sistemas o aplicaciones que puedan ejecutarse desde cualquier computadora, ya que es una de las necesidades primarias de empresas que luchan por ingresar, mantenerse o extenderse en un mercado cada vez más competitivo y agresivo.
El sistema que se presenta en esta tesis soluciona las necesidades de comercialización de equipos de comunicación de una empresa empleando tecnología web, con esto se pretende abarcar nuevos nichos de mercado a los que anteriormente no estaban dirigidos los esfuerzos de la compañía.

Objetivo

El objetivo de este trabajo es presentar de manera teórica y práctica, tanto la metodología como la tecnología más frecuentemente utilizadas para el diseño, desarrollo e implementación de aplicaciones web.

Definición del problema

El problema consiste en desarrollar un sistema que sea de fácil acceso, seguro, con disponibilidad de horario, que controle el flujo de la operación establecida por la empresa para la comercialización de equipos de comunicación.

A continuación se presentan agrupados los requerimientos que se identificaron para el desarrollo del sistema.

- Roles o perfiles de trabajo
    - El sistema deberá contar con roles o perfiles de trabajo que puedan definir las responsabilidades y tareas que los usuarios tendrán dentro del sistema.

- Mantenimiento a catálogos
    - El sistema validará el rol o perfil de trabajo del usuario que se tenga permitido para realizar transacciones en los catálogos del sistema.
    - Permitirá administrar y configurar la información que el sistema necesitará para su correcto funcionamiento.

- Administración del inventario local
    - Permitirá realizar la recepción de las transferencias de los equipos que son enviados desde el almacén central de la empresa. En este punto se validará que el número de serie que se encuentra en el equipo sea igual al número de serie que el almacén central envía al punto de venta.
    - Permitirá realizar la consulta del estado del inventario local del sistema; es decir, será posible visualizar por modelo la cantidad y el detalle de los equipos vendidos, los equipos que se encuentren reservados en alguna venta en tránsito, los equipos disponibles para ventas futuras y los equipos pendientes de verificar el número de serie del equipo con el número de serie del sistema.
    - Permitirá realizar la depuración de los modelos de equipos y sus correspondientes números de serie que ya no serán utilizados en ventas futuras.

- Orden de contratación
    - El sistema validará el rol o perfil de trabajo del usuario que se tenga permitido para realizar transacciones en este módulo del sistema.
    - Validará la existencia del cliente en el sistema por medio de su RFC, si el RFC del cliente no es encontrado en el sistema se permitirá la captura de una orden de contratación, el sistema le asignará de forma automática el identificador *cuenta nueva* y se le dará al usuario la opción de elegir si en la orden de contratación se desglosará el IVA que corresponde a la plaza donde se realiza la venta o si la orden de contratación presentará el IVA incluido. El usuario ingresará la información personal del cliente, la información de los equipos que el cliente desea adquirir, así como la forma mensual de pago del servicio de los equipos, el sistema calculará sobre la base de las reglas del negocio el monto total a cobrar en la orden de contratación.
    - Si el RFC del cliente es encontrado en el sistema se permitirá la captura de una orden de contratación, el sistema le asignará de forma automática el identificador *adición a cuenta* e identificará el esquema del IVA asociado al cliente; es decir, se validará si el IVA debe ir desglosado o incluido en la orden de contratación. El sistema recuperará la información personal del cliente y el usuario especificará la información de los equipos que el cliente desea adquirir, así como la forma mensual de pago del servicio de los equipos, el sistema calculará sobre la base de las reglas del negocio el monto total a cobrar en la orden de contratación.
    - El sistema cambiará de forma automática el estado de *disponible* de los equipos del inventario local que sean asociados a la orden de contratación a un estado de *reservado*. El sistema validará y controlará la concurrencia de usuarios en un mismo punto de venta al momento de generar órdenes de contratación para no permitir que se mezclen números de serie en la generación de las mismas.
    - El sistema validará que encuentre toda la información obligatoria en la orden de contratación y enviará al área de crédito la información ingresada por el usuario para su evaluación y/o investigación del cliente en el Buró de Crédito.

- El sistema permitirá la consulta de la respuesta de la evaluación realizada a la orden de contratación por parte del área de crédito. Si la orden de contratación es rechazada se presentará el motivo del rechazo al usuario y cancelará la solicitud y cambiará automáticamente el estado de los equipos asociados a la orden de contratación de *reservado* a *disponible* para su futuro uso en otra orden de contratación.
- El sistema permitirá la cancelación de una orden de contratación que ya haya sido aprobada por el área de crédito y cambiará automáticamente el estado de los equipos asociados a la orden de contratación de *reservado* a *disponible* para su futuro uso en otra orden de contratación.

- Evaluación de orden de contratación
  - El sistema validará el rol o perfil de trabajo del usuario que se tenga permitido para realizar transacciones en este módulo del sistema.
  - El sistema será capaz de identificar la plaza y punto de venta de origen de la orden de contratación y direccionará la información de la misma a un área específica de crédito para su evaluación.
  - El sistema permitirá al usuario evaluar la información de la orden de contratación, si la solicitud es rechazada, el usuario podrá asignar un motivo de rechazo y enviar la respuesta de la evaluación al punto de venta donde se generó la solicitud.
  - Cuando una solicitud sea rechazada el sistema deberá poner un estado de *rechazada* a la orden de contratación, cancelará la solicitud y cambiará de forma automática el estado de *reservado* de los equipos del inventario local que sean asociados a la orden de contratación a un estado de *disponible* para su futuro uso en otra orden de contratación.

- Asignación de servicios a equipos
  - El sistema validará el rol o perfil de trabajo del usuario que se tenga permitido para realizar transacciones en este módulo del sistema.
  - Cuando la orden de contratación sea aprobada el sistema identificará el plan tarifario que está asociado al equipo y presentará los servicios por omisión que deberá de tener el equipo, así como los servicios opcionales que podrán ser asociados al equipo. El sistema deberá aplicar las validaciones correspondientes según las reglas del negocio para permitir la convivencia de los servicios por omisión y los servicios opcionales.
  - Cuando se haya completado la asociación de servicios a los equipos, el sistema deberá presentar la impresión del contrato de servicios al usuario con toda la información de la orden de contratación capturada.

- Cobranza de orden de contratación
    - El sistema validará el rol o perfil de trabajo del usuario que se tenga permitido para realizar transacciones en este módulo del sistema.
    - El sistema deberá validar la existencia de inicio de operaciones en la caja donde se procederá a recibir el pago de la orden de contratación elaborada.
    - El sistema permitirá la selección de la orden de contratación que se desea cobrar mostrando la información del cliente y el importe total de la orden de contratación.
    - El sistema presentará al usuario las opciones necesarias para identificar y procesar el pago de la orden de contratación realizado por parte del cliente.
    - En el cobro de las órdenes de contratación no se podrán recibir tarjetas de débito, por lo que el sistema deberá validar el acceso de las mismas.
    - El sistema tendrá la capacidad de realizar cargos en línea con el banco cuando el tipo de pago sea con tarjeta de crédito.
    - Cuando el proceso de cobranza haya terminado, el sistema deberá presentar la impresión de un comprobante de pago y la impresión de una factura donde se detalle la forma de pago recibida en el cobro de la orden de contratación y la información de los equipos que han sido adquiridos por el cliente respectivamente.
    - El sistema deberá enviar al proceso de activación la información de los equipos asociados a la orden de contratación para iniciar el proceso de activación de los mismos.
    - El sistema deberá permitir al usuario poder generar el cierre de operaciones de la caja donde se recibieron los pagos de las órdenes de contratación.
    - Cuando una caja esté en operaciones el sistema deberá validar que solamente el usuario que generó el inicio de operaciones pueda accesar a la misma para realizar más operaciones.
    - Cuando se termine el cierre de operaciones de una caja el sistema permitirá que la caja sea seleccionada por cualquier usuario que tenga el rol o perfil de trabajo permitido para hacerlo.

- Contabilidad de operaciones
    - El sistema validará el rol o perfil de trabajo del usuario que se tenga permitido para realizar transacciones en este módulo del sistema.
    - Este proceso solamente se deberá iniciar cuando se realice el cierre de operaciones y tendrá por objetivo registrar los asientos contables de las operaciones que se hicieron desde el último cierre de operaciones.

- Interfaces del sistema
    - El sistema deberá permitir la ejecución de interfaces que le permitan recibir del sistema BSCS (*Business Support and Control System*, el cual es sistema de facturación celular utilizado por la empresa) las secuencias numéricas para la formación de los números de clientes cuando el pago de una orden de contratación sea concluido, también deberá permitir el envío de la información personal del cliente, planes tarifarios y servicios asociados a los equipos para su futura facturación.
    - El sistema deberá permitir la ejecución de interfaces que le permitan enviar la información de las transacciones realizadas durante el día en el inventario de equipos y en la caja del punto de venta a la *Suite de Oracle Financial*, el cual es el ERP (Enterprise Resource Planing) que concentra, organiza y administra toda la información referente a la empresa.

Restricciones

El ambiente donde tiene que instalarse el *Sistema de Comercialización en Línea*, cuenta con las siguientes características:

- Equipo HP-UX 9000 con sistema operativo UNIX.
- Base de datos Oracle, versión 8i.

Solución propuesta

La tecnología que se planteó para el desarrollo del sistema fue Internet, debido a que su uso es de propósito general y no hay limitaciones en cuanto a su uso, además su facilidad de acceso, seguridad y su disponibilidad lo hacen el medio indicado para el funcionamiento del sistema.

El lenguaje de programación que se eligió fue Java porque es un lenguaje orientado a objetos y esto presenta varios beneficios; reutilización de código fuente, seguridad, potencia; Java tiene un gran número de librerías que dan soporte a las diversas necesidades del programador, portabilidad del código; es decir, que el programa puede ser escrito y compilado en una plataforma y ejecutado en otra.

Justificación de la solución

Para que una organización pueda competir en un mercado globalizado debe de cumplir con varias características en el manejo de su información [JOH00], a continuación se mencionan algunas de ellas:

- Consistencia
- Seguridad
- Disponibilidad
  - Rápida y en todo momento.
  - En cualquier lugar.
  - En cualquier plataforma.

El objetivo es poder desarrollar aplicaciones poderosas y flexibles que puedan satisfacer necesidades empresariales que permitan a las organizaciones sobresalir en un mercado competitivo.

Internet

Las nuevas aplicaciones desarrolladas para Internet cubren perfectamente los tres puntos anteriores que buscan las empresas para poder sobresalir en la dinámica de los mercados actuales.

Sabemos que Internet es la conexión de un conjunto de redes locales dispersas [ROB00]. Y cualquier red se puede conectar, siempre y cuando, sea capaz de entender el protocolo TCP/IP. Esto es muy útil ya que se pueden conectar redes a Internet, sin que el funcionamiento interno de la subred se vea afectado; basta con instalar una interfaz entre la red e Internet que se encargue de decidir si un envío de datos es interno a la red y en ese caso enviarlo usando el protocolo propio de la red, o si es a Internet en cuyo caso se traduce al formato propio de Internet.

Para garantizar una entrega rápida de los datos que cruzan Internet, se definen dos niveles. Un primer nivel donde están la mayoría de las subredes y usuarios domésticos, donde el tráfico de datos no es muy elevado y por tanto basta con conexiones de mediana velocidad, el segundo nivel es la verdadera autopista, es una red formada por nodos de conmutación muy rápidos, que recogen el tráfico de las subredes locales y hacen todo lo posible por garantizar una entrega eficaz, a la red formada por estos nodos se le llama red troncal.

Internet es el medio que mejor desempeño ha mostrado en la transmisión de información gracias a sus características, a continuación menciono las siguientes:

- Expansibilidad

  Pueden añadirse, quitarse y sustituirse componentes fácilmente. Además añadir nuevas prestaciones no supone ningún problema ya que TCP/IP no determina qué se puede y qué no se puede transmitir, sino la manera de transmitirlo.

- Universalidad

  Es una red a la que puede accesar cualquier persona.

- Tolerancia a fallos

  El protocolo TCP es capaz de detectar cuando un paquete se ha perdido por el camino, e IP es capaz de detectar cuando un ordenador de Internet tiene una avería o saturación de tráfico y enviar los datos por otras rutas alternativas.

- Integridad

  Internet dispone de mecanismos para asegurar que la información recibida es exacta a la enviada.

Java

Java es un lenguaje de programación de alto nivel con el que se pueden escribir tanto programas convencionales como para Internet [CAN99]. Las características principales que nos ofrece Java respecto a cualquier otro lenguaje de programación son:

- Simplicidad

  Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos.

  Java elimina muchas de las características de otros lenguajes como C++, para mantener reducidas las especificaciones del lenguaje y añadir características muy útiles como el garbage collector (reciclador de memoria dinámica). No es necesario preocuparse de liberar memoria, el reciclador se encarga de ello y como es un thread de baja prioridad, cuando entra en acción, permite liberar bloques de memoria muy grandes, lo que reduce la fragmentación de la memoria.

  Además, el intérprete completo de Java que hay en este momento es muy pequeño, solamente ocupa 215 Kb de RAM.

- Orientación a objetos

  Java implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo. Las plantillas de objetos son llamadas, como en C++, clases y sus copias, instancias. Estas instancias, como en C++, necesitan ser construidas y destruidas en espacios de memoria.

En C++ se suele trabajar con librerías dinámicas (DLLs) que obligan a recompilar la aplicación cuando se retocan las funciones que se encuentran en su interior. Este inconveniente es resuelto por Java mediante una interfaz específica llamada RTTI (RunTime Type Identification) que define la interacción entre objetos excluyendo variables de instancias o implementación de métodos. Las clases en Java tienen una representación en tiempo de ejecución que permite a los programadores interrogar por el tipo de clase y enlazar dinámicamente la clase con el resultado de la búsqueda.

- Independencia de la plataforma

  Una de las ventajas significativas de Java sobre otros lenguajes de programación es que es independiente de la plataforma, tanto en código fuente como en binario. Esto quiere decir que el código producido por el compilador de Java puede transportarse a cualquier plataforma que tenga instalada una máquina virtual Java y ejecutarse. Pensando en Internet esta característica es crucial ya que esta red conecta ordenadores muy distintos.

  Los lenguajes al uso, como C o C++, deben ser compilados para un chip, y si se cambia el chip, todo el software debe compilarse de nuevo por lo tanto esto encarece mucho los desarrollos de los proyectos.

- Seguridad

  Las aplicaciones de Java resultan extremadamente seguras, ya que no accesan a zonas delicadas de memoria o de sistema, con lo cual evitan la interacción de ciertos virus. Java no posee una semántica específica para modificar la pila de programa, la memoria libre o utilizar objetos y métodos de un programa sin los privilegios del kernel del sistema operativo.

  Java no permite abrir ningún archivo en la máquina local (siempre que se realizan operaciones con archivos, éstas trabajan sobre el disco duro de la máquina de donde partió el applet), no permite ejecutar ninguna aplicación nativa de una plataforma e impide que se utilicen otros ordenadores como puente, es decir, nadie puede utilizar nuestra máquina para hacer peticiones o realizar operaciones con otra. Además, los intérpretes que incorporan los navegadores de la web son aún más restrictivos. Bajo estas condiciones (y dentro de la filosofía de que el único ordenador seguro es el que está apagado), se puede considerar que Java es un lenguaje seguro.

- Procesamiento múltiple

  Al ser multihilo, Java es capaz de ejecutar varias actividades simultáneas en un programa. Los hilos, son básicamente pequeños procesos o piezas independientes de un gran proceso. Al estar los hilos construidos en el lenguaje, son más fáciles de usar y más robustos que sus homólogos en C o C++.

  El beneficio de ser multihilo consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real. Aunque el comportamiento en tiempo real está limitado a las capacidades del sistema operativo subyacente (Unix, Windows, etc.), aún supera a los entornos de flujo único de programa (single-threaded) tanto en facilidad de desarrollo como en rendimiento.

CAPÍTULO 2

Análisis del problema

Para el control y desarrollo de proyectos la empresa utiliza una metodología propietaria llamada ISDM (*International System Development Methology*), dicha metodología es una aproximación estructurada para desarrollar, actualizar y comprar sistemas. ISDM esta diseñado para proporcionar todas las partes involucradas en el proceso de desarrollo de sistemas con un conjunto estandarizado de actividades, responsabilidades, plantillas y expectativas para actividades requeridas. La *metodología internacional para el desarrollo de sistemas* además establace un marco de trabajo común para describir, asignar y medir la terminación de tareas requeridas en el desarrollo de un sistema. El ISDM puede ser aplicado a cualquier tamaño y tipo de proyecto. La metodología tiene como finalidad introducir estandares y disciplinas que ayudarán a desarrollar y entregar sistemas de alta calidad.

El ciclo del desarrollo del sistema comienza cuando el usuario especifica sus necesidades a través de un documento llamado *Requerimiento de Negocio*, en este documento el usuario define las reglas del negocio que se deben de tomar en cuenta para el desarrollo del proyecto. Una vez terminado este documento es entregado al área de sistemas, el cual verifica su contenido y realiza un levantamiento de información con el usuario y/o usuarios involucrados, cuando se tiene recabada la información suficiente por parte del área de sistemas éste debe elaborar un documento llamado *Caso de Negocio* utilizando un leguaje no técnico; es decir, se emplea un lenguaje coloquial de tal manera que el usuario pueda entender la solución propuesta por el área de sistemas. Cuando el usuario acepta la solución presentada se concluye de manera formal el *Caso de Negocio*.

Con el documento final del caso de negocio y dado que la tecnología seleccionada para el desarrollo del sistema fue Java comenzé el análisis del sistema utilizando UML (*Unified Modeling Language*) el cual es un lenguaje que permite especificar sistemas que utilizan conceptos orientados a objetos.

Unified Modeling Language, UML

UML (Lenguaje Unificado de Modelado) se define como *un lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software*. Es un sistema notacional (entre otras cosas, incluye el significado de sus notaciones) destinado a los sistemas de modelado que utilizan conceptos orientados a objetos.

Con la creación de UML se consigue obtener un lenguaje que sea capaz de abstraer cualquier tipo de sistema, sea informático o no, mediante diagramas; es decir, mediante representaciones gráficas que contienen toda la información relevante del sistema. Un *diagrama* es una representación gráfica de una colección de elementos del modelo, que habitualmente toma forma de grafo donde los arcos que conectan sus vértices son las relaciones entre los objetos y los vértices se corresponden con los elementos del modelo.

Para obtener más información de UML diríjase al *APÉNDICE A*.

Artefactos para el desarrollo de proyectos

Un *artefacto* es una información que es generada o producida mediante un proceso de desarrollo de software. Pueden ser artefactos un modelo, una descripción o un software. Los artefactos de UML se especifican en forma de diagramas, éstos constituyen los artefactos principales que el modelador puede observar.

Se necesita más de un punto de vista para llegar a representar un sistema. UML utiliza los diagramas gráficos para obtener estos distintos puntos de vista de un sistema:

- Diagramas de casos de uso.
- Diagramas de secuencias.
- Diagramas de clases.

El proceso de desarrollo

UML no define un proceso concreto que determine las fases de desarrollo de un sistema, las empresas pueden utilizar UML como lenguaje para definir sus propios procesos y lo único que tendrán en común con otras organizaciones que utilicen UML serán los tipos de diagramas.

UML es un método independiente del proceso. Los procesos de desarrollo deben ser definidos dentro del contexto donde se van a implementar sistemas [UNLPALM].

Diagramas de casos de uso

Un *diagrama de casos de uso* explica gráficamente un conjunto de casos de uso de un sistema, los actores y la relación de éstos y los casos de uso. Estos últimos se muestran en óvalos y los actores son figuras estilizadas. Hay líneas de comunicación entre los casos y los actores; las flechas indican el flujo de la información o el estímulo.

El diagrama tiene por objeto ofrecer una clase de diagrama contextual que nos permite conocer rápidamente los actores externos de un sistema y las formas básicas que lo utilizan [LARMAN99].

Para obtener más información de diagramas de casos de uso diríjase al *APÉNDICE B*.

Para consultar un ejemplo de diagramas de casos de uso diríjase al *APÉNDICE C*.

Diagramas de secuencias

Los diagramas de secuencias de un sistema se preparan durante la fase de análisis de un ciclo de desarrollo y muestran gráficamente los eventos que fluyen de los actores al sistema. Además se define como la representación que muestra, en determinado escenario de un caso de uso, los eventos generados por actores externos, su orden y los eventos internos del sistema. Su formulación depende de la formulación previa de un caso de uso.

Para obtener más información de diagramas de secuencias diríjase al *APÉNDICE D*.

Para consultar un ejemplo de diagramas de secuencias diríjase al *APÉNDICE E*.

Modelo conceptual

Un modelo conceptual es la representación de objetos en un dominio del problema. En UML, se ilustra con un conjunto de diagramas de estructura estática donde no se define ninguna operación. La designación de un modelo conceptual ofrece la ventaja de subrayar fuertemente la concentración en los conceptos del dominio, no en las entidades del software [LARMAN99].

El modelo conceptual de UML puede mostrarnos

- Conceptos
- Asociaciones entre conceptos
- Atributos de conceptos

Además de descomponer el espacio del problema en unidades comprensibles (conceptos), la creación de un modelo conceptual contribuye a esclarecer la terminología o nomenclatura del dominio. Podemos verlo como un modelo que comunica cuáles son los términos importantes y cómo se relacionan entre sí.

Diagrama de clases

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Normalmente contiene la siguiente información:

- Clases, asociaciones y atributos
- Interfaces, con sus operaciones y constantes
- Métodos
- Información sobre los tipos de los atributos
- Navegabilidad
- Dependencias

A diferencia del modelo conceptual, un diagrama de este tipo contiene las definiciones de las entidades del software en lugar de conceptos del mundo real [LARMAN99].

El diagrama de clases es el diagrama principal para el análisis y diseño. Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia. La definición de clase incluye definiciones para atributos y operaciones. El modelo de casos de uso aporta información para establecer las clases, objetos, atributos y operaciones.

Para obtener más información de diagramas de clases diríjase al *APÉNDICE F*.

Para consultar un ejemplo de diagramas de clases diríjase al *APÉNDICE G*.

CAPÍTULO 3

Diseño del sistema

Las herramientas *CASE* (Computer Aided Software Engineering) o herramientas de ingeniería de software auxiliadas por computadora están diseñadas para automatizar el proceso de análisis de un sistema. Dos de las mejores herramientas *CASE* del mercado son Oracle Designer 2000 y Rational Rose.

Una herramienta *CASE* transforma la información básica que se introduce en ella en entidades, atributos, datos y relaciones entre entidades. Adicionalmente es necesario contar con los conocimientos necesarios de diseño de base de datos y de las reglas de normalización para realizar un buen diseño de un sistema [KEOGH].

Identificación de la información

La información recabada en la generación de los diagramas de UML nos permitió tener una idea más clara para comenzar el diseño de la base de datos relacional que deberá soportar la funcionalidad del *Sistema de Comercialización en Línea*.

El paso inicial al definir un esquema de base de datos es identificar toda la información que utilizará el sistema que se va a desarrollar.

Una base de datos es un conjunto de datos que gestiona un sistema de gestión de base de datos (*database management system o DBMS*). La base de datos comercial sobre la que se desarrollará el sistema es Oracle.

Un esquema de una base de datos es un documento que define todos los componentes de la base de datos, como son las tablas, columnas e índices. Un esquema de una base de datos muestra también las relaciones entre las tablas.

La información que utilizará el sistema se tranforma en entidades. Se define como entidad a cualquier objeto, real o abstracto, que existe en un contexto determinado o puede llegar a existir y del cual deseamos guardar información. Cada entidad se define mediante atributos. Un atributo es la información que define a una entidad.

Una vez identificados los atributos es necesario describir la características de cada una de ellos [KEOGH].

Agrupación de la información

El objetivo de la agrupación de la información es identificar elementos duplicados que puedan ser utilizados en más de una entidad. Los elementos duplicados se deben de eliminar.

El éxito de una base de datos relacional se fundamenta en la existencia de llaves primarias y foráneas en los grupos de datos, y de esta forma crear relaciones entre los grupos. La existencia de esta relación se conoce como integridad referencial.

La integridad referencial permite establecer restricciones en la base de datos. Esto significa que, para asegurar la integridad referencial, el DBMS (*Data base Manager System*) evita la modificación o eliminación de las llaves primarias y foráneas. De la misma forma, las restricciones de la base de datos evitan duplicar filas para mantener la integridad referencial [KEOGH].

Para obtener más información de conceptos de base de datos relacionales diríjase al *APÉNDICE H*.

Diagrama entidad relación

El diagrama Entidad / Relación (E/R) es la representación gráfica de las relaciones entre las entidades que conforman un sistema. Existen dos niveles en el diseño de un E/R, el nivel lógico; donde se definen los nombres de las entidades en singular y se muestran los atributos de cada entidad. El nivel físico del diagrama entidad relación; donde se definen los nombres de las entidades en plural, así como también se define el tipo de dato que va a asociarse a cada atributo y el comportamiento que va a tener dicho atributo; por ejemplo, se define si el atributo va a ser obligatorio u opcional, si va a pertenecer a algún dominio, si se restringirá a que acepte solamente un conjunto de valores determinados, etc. El nivel físico del diagrama E/R será el que finalmente determinará la estructura de la base de datos de un sistema.

Para consultar el diccionario de datos del *Sistema de Comercialización en Línea* diríjase al *APÉNDICE I*.

CAPÍTULO 4

Desarrollo del sistema

Con la fase de análisis concluida, en la cual se terminaron los diagramas de *UML* que muestran las abstracciones del sistema; es decir, las representaciones gráficas que presentan toda la información relevante del sistema y con la fase de diseño terminada en donde de definió la estructura de la base de datos relacional que deberá soportar la funcionalidad del sistema, se tienen las condiciones necesarias para comenzar el desarrollo del sistema, en donde se establece seguir la especificación J2EE, la cual es una arquitectura para implementar aplicaciones empresariales usando Java y tecnologías de Internet.

Una aplicación J2EE se centra alrededor de un conjunto de tecnologías Java claves que proporcionan funcionalidad tanto de cliente como de servidor.

Java 2 Enterprise Edition (J2EE)

La plataforma J2EE utiliza un modelo de aplicación distribuida multicapa. La aplicación lógica es dividida en componentes de acuerdo a su función, y los diversos componentes de aplicación realizados en una aplicación J2EE son instalados en diferentes máquinas dependiendo de la capa a la que pertenecen dentro del ambiente multicapas de J2EE. La figura 4.1 muestra las aplicaciones multicapa de J2EE divididas en las capas descritas en el siguiente listado:

- *Capa – Cliente* se ejecuta en la máquina del cliente.
- *Capa – web* se ejecuta en el servidor J2EE.
- *Capa – Negocios* se ejecuta en el servidor J2EE.
- Capa *Enterprise Information System* (EIS) se ejecuta en el servidor EIS.

Aunque una aplicación J2EE puede consistir de tres o cuatro capas, las aplicaciones multicapa J2EE son generalmente consideradas para ser aplicaciones de tres capas porque ellas son distribuidas sobre tres diferentes instancias: la máquina del cliente, el servidor J2EE, y la base de datos [SUN].

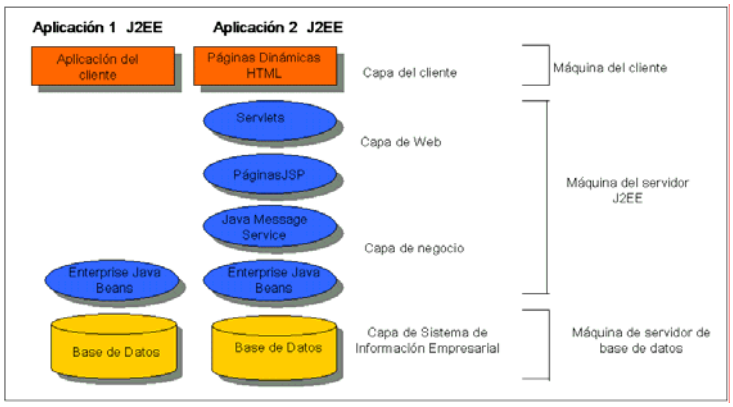


Figura 4.1 Aplicaciones multicapa

Para obtener más información de los componentes J2EE diríjase al *APÉNDICE J*.

Servlets

Anteriormente se mencionó que una aplicación J2EE es un conjunto de componentes, donde cada uno de los cuales proporciona a la aplicación un servicio determinado. La interfaz de usuario es el componente de más alto nivel y más visible, se utiliza para brindar a una persona una forma intuitiva de llamar servicios.

La interfaz de usuario obtiene información de la persona que utiliza la aplicación J2EE y envía una petición a un componente para procesarla. El componente que procesa la petición se construye ya sea como un servlet o como una JavaServer Page.

Un servlet es un programa del servidor que llama la interfaz de usuario u otro componente J2EE y que contiene la lógica del negocio para procesar una petición.

Beneficios de utilizar servlets

La tecnología de servlets evita las ineficiencias. En primer lugar, solamente se carga una copia de un servlet en la máquina virtual Java sin importar el número de peticiones que tenga que atender. Con cada petición se inicia un nuevo hilo en lugar de un nuevo proceso, lo cual reduce el uso de memoria del servidor y reduce el tiempo de respuesta, ya qué solo es necesario tener una copia del servlet en la memoria a la vez. Los servlets por lo general se ejecutan dentro del mismo proceso que el contenedor de servlets. Cada copia del servlet se ejecuta en un hilo.

Un servlet tiene persistencia, lo cual quiere decir que el servlet sigue vivo una vez terminada la petición. Si es necesario por los requerimientos de la aplicación, los datos que utiliza el servlet se pueden mantener en memoria entre una llamada al servlet y la siguiente [KEOGH].

Para obtener más información de Servlets diríjase al *APÉNDICE K*.

Utilización de Servlets en el sistema de comercialización en línea

El uso de servlets en el desarrollo de la aplicación web fue básicamente en procesos de segundo plano (*background*, procesos que son ejecutados directamente en el servidor), esto nos permitió aprovechar las mismas caracteríticas que se mencionaron en el apartado superior. Por ejemplo, se utilizaron servlets en la conexión a la base de datos, en monitoreo de servidores disponibles de JMS, etc.

Para consultar un ejemplo de código fuente de Servlets diríjase al *APÉNDICE L.*

JavaServer Pages

Una JavaServer Page (*JSP*) es un programa del servidor cuyo diseño y funcionalidad son similares a los de un servlet. Un cliente llama a una *JSP* para que le proporcione un servicio web cuya naturaleza depende de la aplicación J2EE. La *JSP* procesa la petición mediante la lógica que incluye o mediante llamadas a otros componentes construidos con tecnología de servlets o Enterprise Java Bean o con alguna otra tecnología. Una vez procesada la petición la *JSP* envía el resultado al cliente.

Sin embargo, la diferencia entre una *JSP* y un servlet radica en la forma en que se escribe la *JSP*. Un servlet se escribe en el lenjuage Java y las respuestas se codifican como objetos String que se envían al método pintnln(). El objeto String debe tener un formato HTML, XML o cualquier otro formato que requiera el cliente.

A diferencia de esto, una *JSP* se escribe en HTML, XML o en el formato del cliente, entremezclado con elementos de código, directivas y acciones escritas en el lenguaje Java y con sintáxis de *JSP* [KEOGH].

Crear una *JSP* es más sencillo que crear un servlet debido a que la *JSP* está escrita en HTML en lugar de Java. Esto significa que una *JSP* no está llena de los métodos println() que se encuentran en un servlet. Sin embargo, una *JSP* proporciona fundamentalmente las mismas características que un servlet, ya que una *JSP* se convierte en un servlet la primera vez que un cliente la solicita.

Para obtener más información de Java Server Pages diríjase al *APÉNDICE M*.

Utilización de JavaServer Pages en el sistema de comercialización en línea

El uso de la tecnología JSP en el desarrollo del sistema fue de gran utilidad sobre todo en las páginas donde se requería un contenido dinámico para manipular y/o presentar la información requerida por el usuario. Por ejemplo, se utilizó JSP para el desarrollo de los catálogos, en la recepción de equipos en el inventario local, en la captura y evaluación de una orden de contratación, en la asignación de servicios a los equipos asociados en una orden de contratación, en la cobranza de la orden de contratación, en la página lanzadora de reportes del sistema, etc.

Se desarrollaron *store procedures* (procedimientos almacenados en la base de datos) para integrar la lógica del negocio de cada sección del sistema, con el objetivo de que el JSP solamente envíe la información necesaria al *store procedures* y éste regrese la respuesta de la petición a su correspondiente JSP, logrando con esto disminuir el tiempo de respuesta en el funcionamiento de las páginas y optimizar el número de conexiones a la base de datos.

Para consultar código fuente de ejemplo de Java Server Pages diríjase al *APÉNDICE N*.

Enterprise JavaBeans (EJB)

La arquitectura J2EE se conforma de componentes que en conjunto permiten a los desarrolladores construir aplicaciones J2EE robustas que aprovechan las eficiencias de la tecnología distribuida.

Por lo general una aplicación J2EE utiliza una interfaz de usuario basada en un navegador y que consta de una página web. Un formulario obtiene información del usuario y envía una petición de un servicio, el cual se envía a un componente del servidor. El componente del servidor puede ser un servlet o una *JSP*.

Al ser intermediarios entre un cliente y los servicios, los servlets y las *JSP* reciben peticiones del servicio de un cliente y las atienden, para lo cual llaman a otros componentes del servidor que contienen la lógica de negocio necesaria para atender la petición de forma parcial o total. El resultado se devuelve al servlet o *JSP*, que responde al cliente.

Muchos de los componentes de servidor que se llaman desde un servlet o *JSP* están escritos en tecnología de EJB [KEOGH].

Un Enterprise JavaBean (*EJB*) es un componente de la arquitectura J2EE cuya función principal es de proporcionar la lógica del negocio de una aplicación J2EE e interacturar con otros componentes J2EE del servidor. La naturaleza de la lógica de negocio y las interacciones con otros componentes J2EE del servidor dependen de la aplicación J2EE.

Un *EJB* se escribe en el lenguaje Java. Esto significa que un *EJB* es independiente del sistema operativo y de la plataforma, siempre y cuando no se utilice código nativo. El desarrollador de un *EJB* se centra en escribir la lógica de negocio para atender las peticiones de servicio y no se preocupa de los servicios de los sistemas a menos que la especificación del *EJB* así lo requiera. El servidor de *EJB* gestiona los servicios del sistema como la gestión de hilos, la seguridad, las transacciones y la persistencia si el *EJB* indica al servidor cómo gestionarlos, o bien el *EJB* puede gestionarlos él mismo [KEOGH].

Para obtener más información de Enterprise JavaBeans diríjase al *APÉNDICE O*.

Utilización de Enterprise Java Beans en el sistema de comercialización en línea

Básicamente se utlizaron Enterprise Java Beans en dos secciones del sistema, en la captura de información y en la cobranza de una orden de contratación ya que estos dos apartados son los más complejos del *Sistema de Comercialización en Línea*. El uso del EJB nos permitió concentrar y estructurar la lógica del negocio de estas dos secciones en componentes independientes, de tal manera que al momento de invocarlos desde su correspondiente JSP, el EJB procesa la información recibida y el resultado de la validación de la información si es exitosa se envía al API de Java Message Service para la evaluación o la activación de los equipos de una orden de contratación respectivamente. De lo contrario si existió algún error el EJB regresa el mensaje de error a la JSP que invocó para que lo muestre al usuario. Los componentes EJB que se desarrollaron en el sistema fueron del tipo sesión.

Se desarrollaron *store procedures* (procedimientos almacenados en la base de datos) para integrar la lógica del negocio y robustecer el desempeño del componente EJB, logrando con esto disminuir el tiempo de respuesta en el procesamiento de las solicitudes al componente y optimizar el número de conexiones a la base de datos.

Para consultar el código fuente de ejemplo de un Enterprise JavaBeans diríjase al *APÉNDICE P.*

Java Message Service (JMS)

Como se comentó en la sección anterior, un componente contiene lógica de negocio para realizar una tarea que puede requerir múltiples aplicaciones.

Sin embargo, los componentes son independientes de las aplicaciones. Un componente aplica lógica del negocio a la información que recibe de una aplicación y devuelve a ésta el resultado de su procesamiento.

El servicio de mensajería de Java es el complemento que conecta las aplicaciones J2EE con los componentes. Proporciona un medio para transmitir mensajes entre ellos.

Fundamentos de JMS

Java Message Service se compone de cinco elementos: un proveedor, clientes, mensajes, objetos administrados y clientes nativos. Un proveedor es el agente de mensajería responsable de gestionar el servicio de mensajería. Los clientes son las aplicaciones y los componentes escritos en Java que utilizan el proveedor para comunicarse entre sí. Los mensajes son objetos que se transmiten entre los clientes. Los objetos administrados son objetos JMS que se utilizan durante la transmisión. Los clientes nativos son las aplicaciones construidas antes de la presentación de JMS que utilizan la API cliente nativa del sistema de mensajería [KOUGH].

Para obtener más información de Java Message Service diríjase al *APÉNDICE Q*.

Utilización de Java Message Services en el sistema de comercialización en línea

El modelo utilizado de JMS en el desarrollo del *Sistema de Comercialización en Línea* fue Point-to-Point, debido a la necesidad de que un mensaje con la información de una orden de contratación llegue a un destinatario especifico y no a un conjunto de destinatarios (modelo publish / subscriber).

Fundamentalmente se utilizó JMS en dos secciones del sistema, en la evaluación de una orden de contratación y en la activación de los equipos asociados a una orden de contratación cobrada. Se utilizó una arquitectura punto a punto donde el cliente envía el mensaje a una cola, la cual esta asociada a un cliente determinado. El mensaje se almacena en la cola hasta que el destinatario se conecta para recuperarlo.

En la evaluación de las órdenes de contratación se tiene un conjunto de operadores que reciben y revisan la información de las órdenes de contratación, consultan el historial crediticio del cliente en el Buró de Crédito y en base a la calificación obtenida por el cliente se aprueba o rechaza la solicitud recibida.

En la activación de equipos se tiene un conjunto de operadores que reciben la información de las órdenes de contratación cobradas, con la información del cliente y de los números de serie de los equipos y los servicios asociados a ellos y realizan las activación de los mismos.

En ambos casos el sistema detecta qué operador está disponible para recibir un mensaje, el API de JMS envía el mensaje y el estado del operador cambia de disponible a ocupado. La respuesta es envíada de regreso al usuario que generó la solicitud.

Para consultar código fuente de ejemplo de Java Message Service diríjase al *APÉNDICE R*.

CAPÍTULO 5

Conclusiones

La formación académica y la disciplina que adquirí en la Facultad de Ingeniería han sido fundamentales para mi desarrollo profesional, las materias que cursé en mi plan de estudios me dieron la capacidad de razonamiento lógico para resolver problemas, la cual es la misión de un ingeniero.

Materias como estructura de datos, estructuras discretas, programación de sistemas, sistemas operativos, compiladores, bases de datos y temas especiales de computación entre otras, me dieron conocimientos que me han permitido poner en práctica en el ambiente laboral y así poder competir y sobresalir en un mercado cada vez más agresivo y complejo, el cual demanda que se tengan sólidos conocimientos sobre el área para enfrentar los retos que las nuevas tecnologías imponen a las empresas en ésta era de la globalización en los negocios.

La iniciativa propia de resolver problemas y de aprender nuevas formas o técnicas de hacer la cosas han sido muy importantes en mi vida profesional ya que es importante estar actualizado en las caracteríticas o novedades que presentan las diferentes tecnologías que son utilizadas por muchas empresas por que me han permitido participar en las toma de decisiones al momento de elegir una opción o de proponer una solución que debe de satisfacer una necesidad o de resolver un problema.

La experiencia adquirida durante mi participación en anteriores proyectos, en los cuales desempeñé diferentes roles o perfiles de trabajo, me ha proporcionado la madurez para para poder interactuar con diferentes personas y desenvolverme para lograr objetivos. Durante el desarrollo de este proyecto tuve la oportunidad de ejercer varias responsabilidades, los cuales me permitieron crecer de forma personal y profesional, a continuación menciono de forma detallada cada uno de ellas.

- **Analista de Sistemas**

  La empresa para la que se desarrolló este sistema tiene una metodología propietaria para el control de las diferentes etapas del desarrollo de un sistema. Mi trabajo consistió en analizar el documento de *caso de negocio* donde en un lenguaje no técnico se detalló el objetivo y alcance del proyecto, el cual fue el resultado de una serie de entrevistas con los usuarios de las áreas involucradas. A partir de este documento empecé a diseñar la solución que satisfízo las necesidades del requerimiento. Anteriormente había tenido varias experiencias sobre el análisis de requerimientos; sin embargo, ésta fue mi primera oportunidad para proponer una solución integral para el desarrollo un proyecto. Con el tiempo aprendí muchas cosas adicionales a cerca de las reglas del negocio de la empresa, las cuales me han permitido participar en desarrollo de más proyectos dentro de la compañía.

- **Diseñador de Base de Datos**

  Con base a los conocimientos adquiridos en la etapa de análisis del proyecto pude identificar las entidades necesarias para el diseño de la base de datos relacional que debe soportar el funcionamiento del sistema. Al igual que en el perfil de analista, ya había tenido participaciones en proyectos donde fue necesario realizar el diseño parcial de una base de datos para dar solución los requerimientos entregados por los usuarios; sin embargo, también ésta fue mi primera oportunidad en realizar el diseño completo de una base de datos relacional para satisfacer las complejas necesidades del proyecto. A partir de esta experiencia puede incrementar mis habilidades para el diseño de bases de datos relacionales, lo cual me ha permitido participar en desarrollo de más proyectos dentro de la empresa.

- **Programador**

  Con la experiencia adquirida en la participación de anteriores proyectos, tuve la oportunidad de retroalimentarme y compartir mis conocimientos con mis compañeros de trabajo, así como de aprender nuevas tecnologías, lenguajes de programación, técnicas o habilidades para el desarrollo de programas más complejos y robustos tanto a nivel de *store procedures* como de lenguajes de programación, etc.

- **Líder de Proyecto**

  Tal vez esta fue el perfil más excitante de todos, ya que aquí intervienen varios factores además del técnico, tales como el aspecto humano, el liderazgo, la motivación, el reconocimiento, la definición objetivos, el trabajo en equipo, delegar responsabilidades y el manejo de trabajo bajo presión, entre otros. Con el aspecto técnico y humano tuve la oportunidad de entrevistar y evaluar a los candidatos para formar el equipo de trabajo para el desarrollo del proyecto, identificando el perfil de cada elemento dentro del equipo. Con el liderazgo, la motivación y el reconocimiento puede identificar puntos de mejora dentro del equipo de trabajo para desempeñar con mayor calidad las actividades especificadas en el plan de trabajo. La definición de objetivos, delegar responsabilidades y el manejo del trabajo bajo presión fueron fundamentales en los momentos críticos del proyecto, ya que fue necesario mantener la ecuanimidad para resolver problemas no detectados y lograr una mejor organización para el desempeño de cada elemento dentro del desarrollo del proyecto.

Definitivamente creo que es importante aprender nuevas cosas cada día y creo que puedo alcanzar mejores resultados si propicio el trabajo en equipo y hago que mi entorno mejore, la forma que tengo para que esto pase es retroalimentarme de las experiencias de otras personas para enriquecerme a mi mismo y compartir mis concocimientos con los demás.

Para consultar la bibliografía recomendada para este capítulo diríjase al *APÉNDICE U*.

APÉNDICE A

Unified Modeling Language, UML

UML (Lenguaje Unificado de Modelado) se define como *un lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software*. Es un sistema notacional (entre otras cosas, incluye el significado de sus notaciones) destinado a los sistemas de modelado que utilizan conceptos orientados a objetos.

El inicio

Nació en 1994 por iniciativa de Grady Booch y Jim Rumbaugh para combinar sus dos famosos métodos: el de Booch y el OMT (Object Modeling Technique, Técnica de Modelado de Objetos). Más tarde se les unió Ivar Jacobson, creador del método OOSE (Object-Oriented Software Engineering, Ingeniería de Software Orientada a Objetos). En respuesta a una petición de OMG (Object Management Group, asociación para fijar los estándares de la industria) para definir un lenguaje y una notación estándar del lenguaje de construcción de modelos, en 1997 propusieron el UML como candidato[LAMAN99] .

Modelado de Objetos

En la especificación del UML podemos comprobar que una de las partes que lo componen es un *metamodelo* formal. Un *metamodelo* es un modelo que define el lenguaje para expresar otros modelos. Un modelo en Orientación a Objetos es una abstracción cerrada semánticamente de un sistema y un sistema es una colección de unidades conectadas que son organizadas para realizar un propósito específico. Un sistema puede ser descrito por uno o más modelos, posiblemente desde distintos puntos de vista.

Una parte de UML define, una abstracción con significado de un lenguaje para expresar otros modelos (es decir, otras abstracciones de un sistema, o conjunto de unidades que se organizan para conseguir un propósito).

UML es una técnica de modelado de objetos y como tal supone una abstracción de un sistema para llegar a construirlo en términos concretos. El modelado no es más que la construcción de un modelo a partir de una especificación. Un modelo es una abstracción de algo, que se elabora para comprender ese algo antes de construirlo. El modelo omite detalles que no resultan esenciales para la comprensión original y por lo tanto facilita dicha comprensión.

La OMT (Object Modeling Technique), por ejemplo, intenta abstraer la realidad utilizando tres clases de modelos Orientados a Objetos: el *modelo de objetos*, que describe la estructura estática; el *modelo dinámico*, con el que describe las relaciones temporales entre objetos; y el *modelo funcional* que describe las relaciones funcionales entre valores. Mediante estas tres fases de construcción de un modelo, se consigue una abstracción de la realidad que tiene en sí misma información sobre las funciones principales de ésta.

Los modelos al no ser una representación que incluya todos los detalles de los originales, permiten probar más fácilmente los sistemas que modelan y determinar los errores. Según se indica en la metodología OMT, los modelos permiten una mejor comunicación con el cliente por distintas razones:

- Es posible enseñar al cliente una posible aproximación de lo que será el producto final.
- Proporcionan una primera aproximación al problema que permite visualizar cómo quedará el resultado.
- Reducen la complejidad del original en subconjuntos que son fácilmente tratables por separado.

Se consigue un modelo completo de la realidad cuando el modelo captura los aspectos importantes del problema y omite el resto.

APÉNDICE B

Casos de uso

El *caso de uso* es un documento narrativo que describe la secuencia de los eventos de un actor (agente externo) que utiliza un sistema para completar un proceso. Un proceso describe, de comienzo a fin, una secuencia de los eventos, de las acciones y las transacciones que se requieren para producir u obtener algo de valor. Los casos de uso son historias o casos de utilización de un sistema; no son exactamente los requerimientos ni las especificaciones funcionales, sino que ejemplifican e incluyen tácitamente los requerimientos en las historias que narran.

Caso de uso de alto nivel

Un caso de uso de alto nivel describe un proceso brevemente, casi siempre en dos o tres enunciados. Conviene servirse de este tipo de caso de uso durante el examen inicial de los requerimientos y del proyecto, a fin de entender rápidamente el grado de complejidad y funcionalidad del sistema. Estos casos son muy breves y vagos en las decisiones del sistema.

Caso expandido de uso

Un *caso expandido* de uso muestra más detalles que uno de alto nivel; este tipo de casos suelen ser más útiles para alcanzar un conocimiento más profundo de los procesos y de los requerimientos. Conviene escribir en el formato expandido los casos más importantes y de mayor influencia [LARMAN99].

Casos de uso primarios, secundarios y opcionales

Los casos deberán clasificarse en primarios, secundarios y opcionales. A partir de estas designaciones se establecerán las prioridades en su desarrollo.

- Los casos primarios de uso representan los procesos comunes más importantes.
- Los casos secundarios de uso representan procesos menores.
- Los casos opcionales de uso representan procesos que pueden no abordarse.

Explicación del formato expandido

La parte superior de la forma expandida es información muy breve.

Caso de Uso	Nombre del caso de uso
Actores	Lista de actores (agentes externos), en la cual se indica quién inicia el caso de uso.
Propósito	Intención del caso de uso
Resumen	Repetición del caso de uso de alto nivel o alguna síntesis similar.
Tipo	1. Primario, secundario u opcional (a explicar) . 2. Escencial o real (a explicar).
Referencias cruzadas	Casos relacionados de uso y funciones también relacionadas del sistema.

La sección intermedia, *curso normal de los eventos*, es la parte medular del formato expandido; describe los detalles de la conversión interactiva entre los actores y el sistema. Un aspecto esencial de la sección es que explica la secuencia más común de los eventos: la historia normal de las actividades y terminación exitosa de un proceso. No incluye situaciones alternas.

Curso normal de los eventos	
Acción del actor	**Respuesta del sistema**
Acciones numeradas de los actores.	Descripciones numeradas de las respuestas del sistema.

La última sección, *curso alterno de los eventos*, describe importantes opciones o excepciones que pueden presentarse en relación con el curso normal. Si son complejas, podemos expandirlas y convertirlas en nuestros casos de uso.

Cursos alternos

Alternativas que pueden ocurrir en el número de línea. Descripción de excepciones.

APÉNDICE C

Ejemplo de caso de uso

El siguiente caso de uso muestra el flujo de los eventos en la generación de una orden de contratación de un cliente nuevo en el *Sistema de Comercialización en Línea*.

Caso de uso de la generación de una orden de contratación con un cliente nuevo

Sección: Principal

Caso de Uso	Generar Orden de Contratación
Actores	Ejecutivo del Punto de Venta (Iniciador).
Propósito	Capturar una orden de contratación
Resumen	Elaboración de una orden de contratación con un cliente nuevo.
Tipo	Primario y esencial.
Referencias cruzadas	

Curso normal de los eventos			
Acción del actor		**Respuesta del sistema**	
1.	Este caso de uso comienza cuando el cliente llega a un punto de venta a comprar equipos.		
2.	El ejecutivo captura en la página el RFC del cliente.	3.	Valida la existencia del cliente en el sistema.
		4.	El sistema manda mensaje ***Cliente no encontrado***.
		5.	El sistema selecciona automáticamente la opción ***Cliente nuevo***.
		6.	El sistema identifica el IVA de la plaza donde se realizará la venta.
		7.	El sistema presenta una lista con el monto y tipo de IVA a aplicar en la orden de contratación ***15% IVA no desglosado*** o ***15% IVA desglosado***. Esta información es obligatoria.
8.	El ejecutivo selecciona el tipo de IVA a aplicar.		
		9.	El sistema despliega la página para accesar los datos personales del cliente. Los datos de esta página son obligatorios.

10.	El ejecutivo ingresa los datos personales del cliente.		
		11.	El sistema presenta la página para accesar el domicilio del cliente. Los datos de esta página son obligatorios.
12.	El ejecutivo ingresa el código postal del cliente.		
		13.	El sistema presenta la lista de colonias que corresponden al código postal. Esta información es obligatoria.
14.	El ejecutivo selecciona la colonia de la lista de colonias.		
		15.	El sistema presenta la página para accesar las referencias comerciales. Esta información es opcional.
		16.	El sistema presenta la página seleccionar la forma de pago mensual del servicio. La información de esta página es obligatoria.
		17.	El sistema presenta la lista de planes contractuales.
18.	El ejecutivo selecciona plan contractual e ingresa el número de meses para plazo del plan contractual.		
19.	El ejecutivo marca si se realizará el cargo recurrente mensual del servicio en la tarjeta de crédito del cliente.		
		20.	El sistema presenta la lista de monedas para efectuar el pago mensual del servicio.
21.	El ejecutivo selecciona la moneda de la lista.		
		22.	El sistema presenta la lista de formas de pago del servicio.
23.	El ejecutivo selecciona la forma de pago ***Efectivo*** cuando no se indique un cargo recurrente.		
24.	El ejecutivo selecciona la forma de pago ***Tarjeta de Crédito*** cuando el campo de cargo recurrente se encuentre marcado.		
		25.	El sistema presenta la lista de bancos cuando la forma de pago sea Tarjeta de Crédito.
26.	El ejecutivo ingresa el número de tarjeta de crédito.		

27.	El ejecutivo ingresa la fecha de expedición de la tarjeta de crédito.		
28.	El ejecutivo ingresa la fecha de expiración de la tarjeta de crédito.		
29.	El ejecutivo ingresa observaciones. Esta información es opcional.		
		30.	El sistema presenta la página de selección de paquetes, planes tarifarios y números de serie.
31.	El ejecutivo oprime botón ***Agregar Paquete***.		
		32.	El sistema filtra los paquetes de acuerdo al tipo de IVA desglosado o no desglosado y a la ciudad donde se efectua la venta.
		33.	El sistema muestra la lista de paquetes disponibles.
34.	El ejecutivo selecciona el paquete deseado.		
		35.	El sistema identifica el número de equipos y modelos que componen al paquete.
		36.	El sistema valida las existencias de los equipos en el inventario local según el modelo de los equipos.
		37.	El sistema presenta la lista de números de serie.
38.	El ejecutivo selecciona el número de serie.		
		39.	El sistema filtra los planes tarifarios según la ciudad donde se efectua la venta.
		40.	El sistema presenta la lista de planes tarifarios.
41.	El ejecutivo selecciona el plan tarifario deseado por el cliente.		
		42.	El sistema calcula el monto del depósito en garantía cuando no se indique un cargo recurrente.
		43.	El sistema calcula el importe de la orden de contratación.
		44.	El sistema calcula el IVA de la orden de contratación.
		45.	El sistema calcula el importe total de la orden de contratación.
		46.	El sistema presenta la información completa de la orden de contratación.

		47.	El sistema presenta mensaje de confirmación de información.
48.	El ejecutivo oprime el botón ***Enviar Información***.		
		49.	El sistema guarda la información en la base de datos.
		50.	El sistema envía la información de la orden de contratación al área de Credito para su evaluación.
		51.	El sistema limpia la información de las páginas.
		52.	El sistema regresa a la página de validación de clientes.

APÉNDICE D

Diagramas de secuencias

Los diagramas de secuencias de un sistema se preparan durante la fase de análisis de un ciclo de desarrollo y muestran gráficamente los eventos que fluyen de los actores al sistema. Además se define como la representación que muestra, en determinado escenario de un caso de uso, los eventos generados por actores externos, su orden y los eventos internos del sistema. Su formulación depende de la formulación previa de un caso de uso.

El ***evento de un sistema*** es un hecho externo de entrada que un actor produce en un sistema. El evento da origen a una operación de respuesta. La ***operación de un sistema*** es una acción que éste ejecuta en respuesta a un evento del sistema.

Para identificar los eventos de un sistema es necesario tener una idea clara de qué se quiere al escoger su frontera [LARMAN99].

Pasos para generar un diagrama de secuencia

- Identifique los actores que operan directamente sobre el sistema.
- A partir del curso normal de los eventos del caso de uso identifique los eventos (externos) del sistema que son generados por los actores. Muéstrelos gráficamente en el diagrama.
- A la izquierda del diagrama de secuencia puede incluir o no el texto del caso de uso.

Interpretación de un diagrama de secuencia

Un diagrama de secuencia posee dos dimensiones: la vertical representa el tiempo, la horizontal representa los objetos que participan en la interacción. En general, el tiempo avanza hacia abajo dentro de la página (se pueden invertir los ejes si se desea). Con frecuencia sólo son importantes las secuencias de mensajes pero en aplicaciones de tiempo real el eje temporal puede ser una métrica. La ordenación horizontal de los objetos no tiene ningún significado.

Cada objeto representa una columna distinta, se pone un simbolo de objeto al final de la flecha que representa el mensaje que ha creado el objeto; está situada en el punto vertical que denota el instante en que se crea el objeto. Esta se conoce como línea de vida del objeto. Se pone una X grande en el punto en que deja de existir el objeto o en el punto en que el objeto se destruye a sí mismo. Para el periodo durante el cual esté activo el objeto, la línea de vida se amplía para ser una línea doble continua. Si el objeto se llama a sí mismo, entonces se superpone otra copia de la doble línea para mostrar la doble activación. El orden relativo de los objetos no tiene significado aún cuando resulta útil organizarlos de modo que se minimice la distancia de las flechas.

Cada mensaje se representa mediante una flecha horizontal que va desde la línea de vida del objeto que envió el mensaje hasta la línea de vida del objeto que ha recibido el mensaje. Si un mensaje requiere un cierto tiempo para llegar a su destino, entonces la flecha del mensaje se dibuja diagonalmente hacia abajo.

Para un flujo de objeto asíncrono entre objetos activos, los objetos se representan mediante líneas dobles continuas y los mensajes se representan como flechas. Se pueden enviar simultáneamente dos mensajes pero no se pueden recibir simultáneamente porque no se puede garantizar una recepción simultánea.

Las bifurcaciones se muestran partiendo la línea de vida del objeto. Cada bifurcación puede enviar y recibir mensajes. Eventualmente las líneas de vida del objeto tienen que fusionarse de nuevo [CREANGEL].

Ejemplo de diagrama de secuencia

Los siguientes diagramas de secuencia muestran el flujo de los eventos en la generación de una orden de contratación con un cliente nuevo de acuerdo al caso de uso presentado en el apéndice C.

Diagrama 1. Presenta el inicio de una orden de contratación para un cliente nuevo.

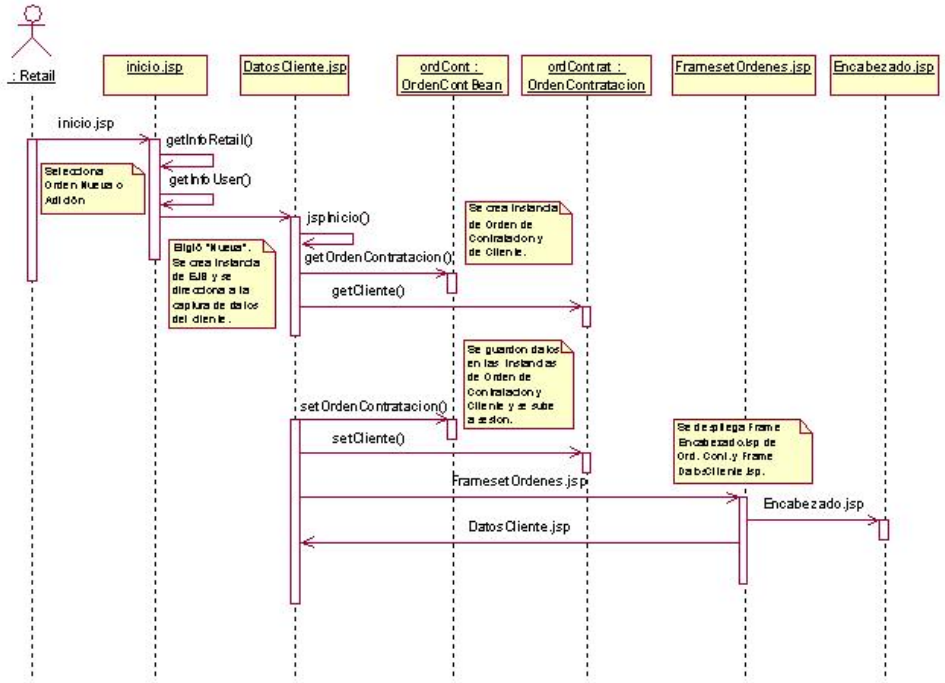


Diagrama 2. Captura de Domicilio Fiscal y de Entrega

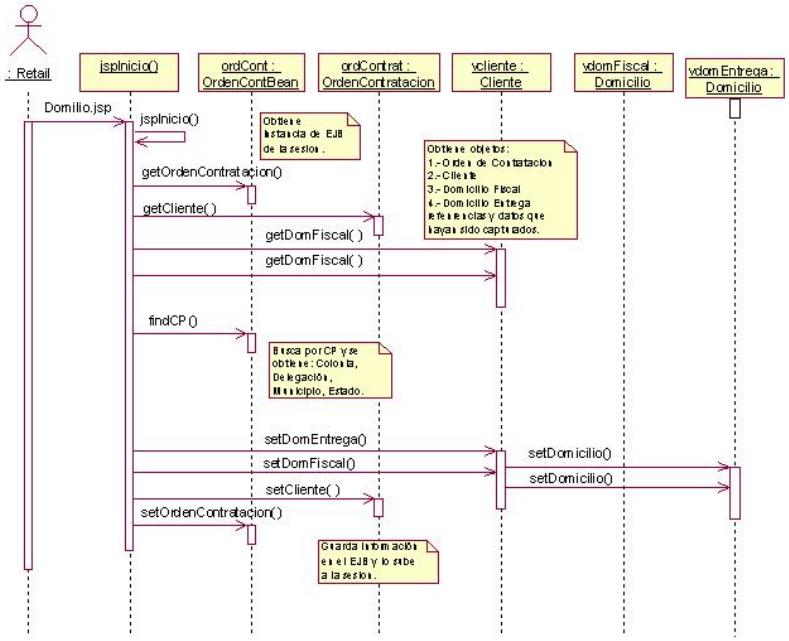


Diagrama 3. Captura de Forma de Pago

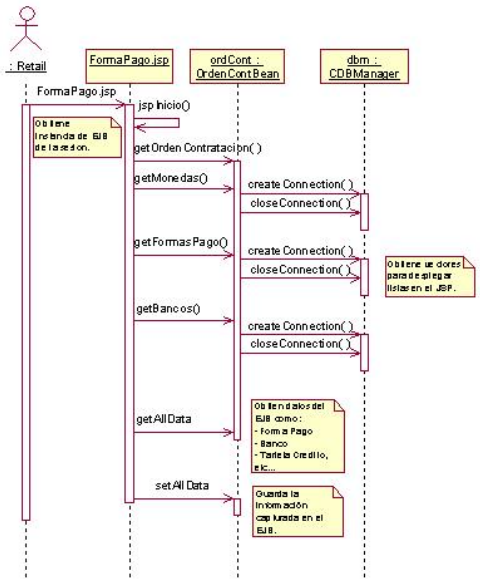
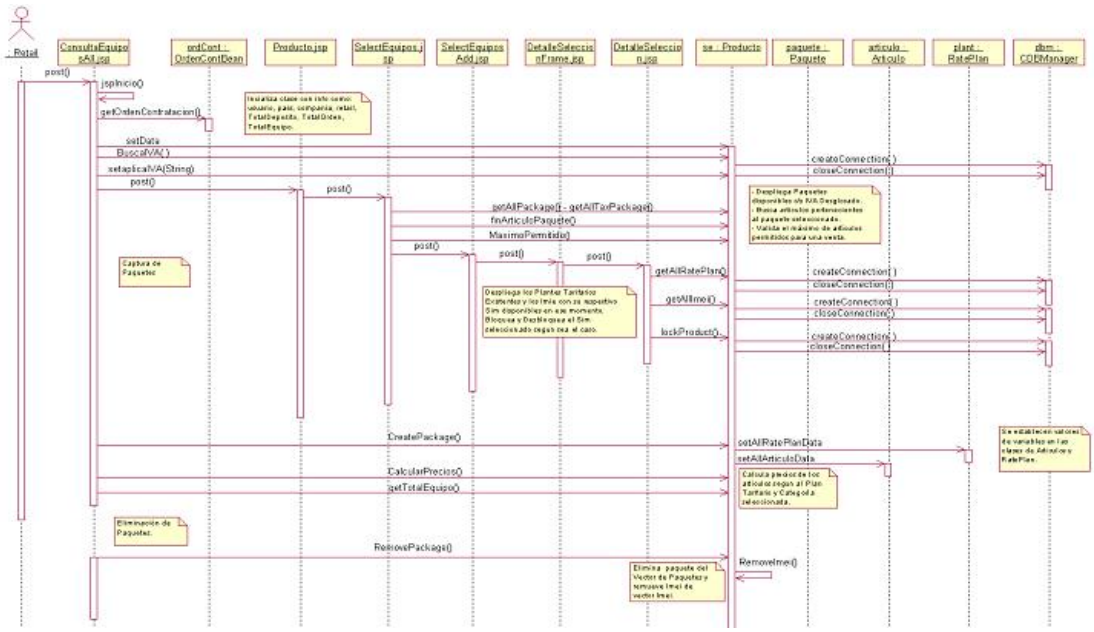


Diagrama 4. Selección de Paquetes de Equipos



APÉNDICE F

Diagrama de clases

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Normalmente contiene la siguiente información:

- Clases, asociaciones y atributos
- Interfaces, con sus operaciones y constantes
- Métodos
- Información sobre los tipos de los atributos
- Navegabilidad
- Dependencias

A diferencia del modelo conceptual, un diagrama de este tipo contiene las definiciones de las entidades del software en lugar de conceptos del mundo real [LARMAN99].

El diagrama de clases es el diagrama principal para el análisis y diseño. Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia. La definición de clase incluye definiciones para atributos y operaciones. El modelo de casos de uso aporta información para establecer las clases, objetos, atributos y operaciones.

Mecanismos de abstracción

- Clasificación / Instanciación
- Composición / Descomposición
- Agrupación / Individualización
- Especialización / Generalización

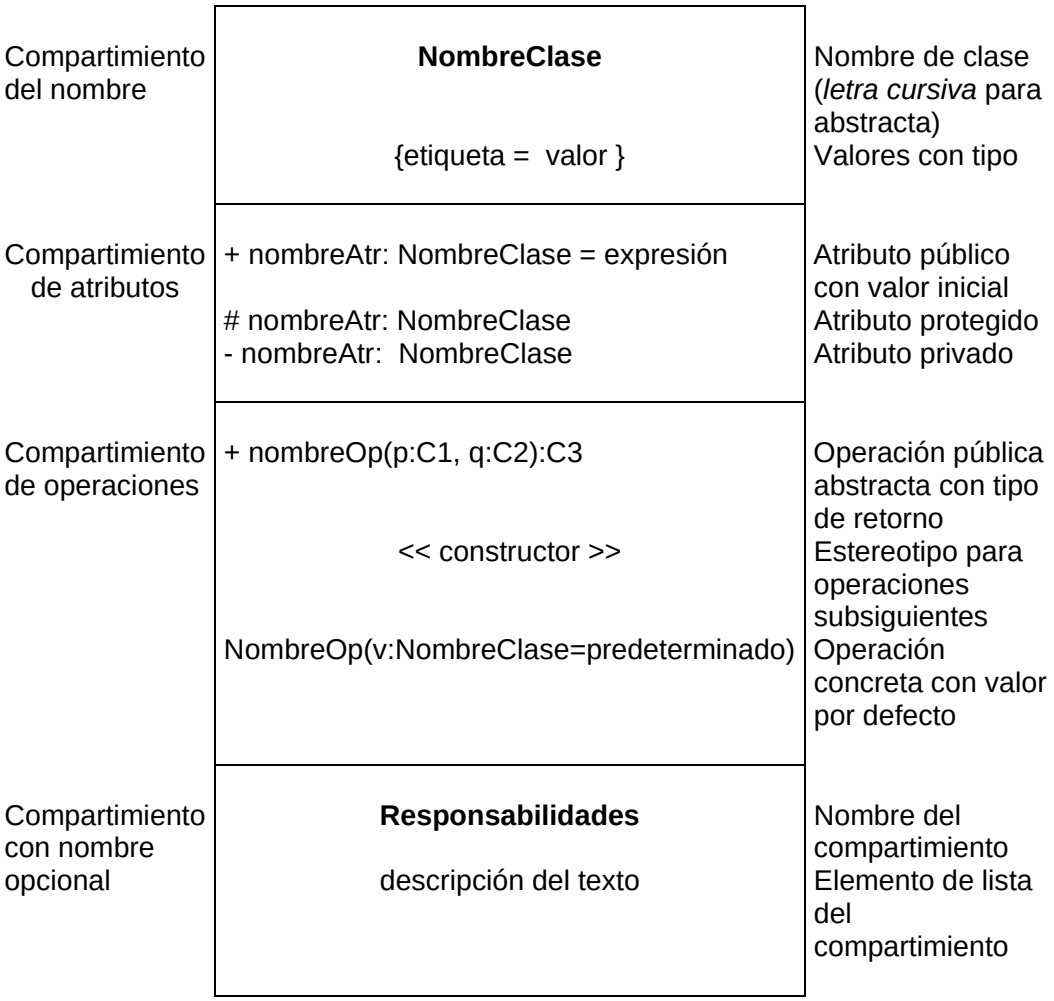
La clasificación es uno de los mecanismos de abstracción más utilizados. La clase define el ámbito de definición de un conjunto de objetos, y cada objeto pertenece a una clase, los objetos se crean por instanciación de las clases.

Cada clase se representa en un rectángulo con tres compartimientos:

- nombre de la clase
- atributos de la clase
- operaciones de la clase

Los atributos de una clase no deberían ser manipulables directamente por el resto de objetos. Por esta razón se crearon niveles de visibilidad para los elementos que son:

NIVEL	SIGNIFICADO	DESCRIPCIÓN
-	Privado	Cada parte es totalmente invisible.
#	Protegido	Los atributos / operaciones están visibles para las clases friends y para las clases derivadas de la original.
+	Público	Los atributos / operaciones están visibles a otras clases.



Relaciones entre clases

Los enlaces entre objetos pueden representarse entre las respectivas clases y sus formas de relación son:

- Asociación y Agregación (vista como un caso particular de asociación)
- Generalización / Especialización.

Las relaciones de Agregación y Generalización forman jerarquías de clases.

Asociación

La asociación expresa una conexión bidireccional entre objetos. Una asociación es una abstracción de la relación existente en los enlaces entre los objetos. Puede determinarse por la especificación de multiplicidad (mínima...máxima)

MULTIPLICIDAD	DESCRIPCION
1	Uno y sólo uno
0..1	Cero o uno
M..N	Desde M hasta N (enteros naturales)
*	Muchos
0..*	Cero o muchos
1..*	Uno o muchos (al menos uno)

Agregación

La agregación representa una relación que parte de entre objetos. En UML se proporciona una escasa caracterización de la agregación. Esta relación puede ser caracterizada con precisión determinando las relaciones de comportamiento y estructura que existen entre el objeto agregado y cada uno de sus objetos componentes.

Una agregación se podría caracterizar según:

¿Puede el objeto parte comunicarse directamente con objetos externos al objeto agregado?

Comunicación Directa	Descripción
No	Inclusiva
Si	No Inclusiva

¿Puede cambiar la composición del objeto agregado?

Cambiar Composición	Descripción
No	Estática
Si	Dinámica

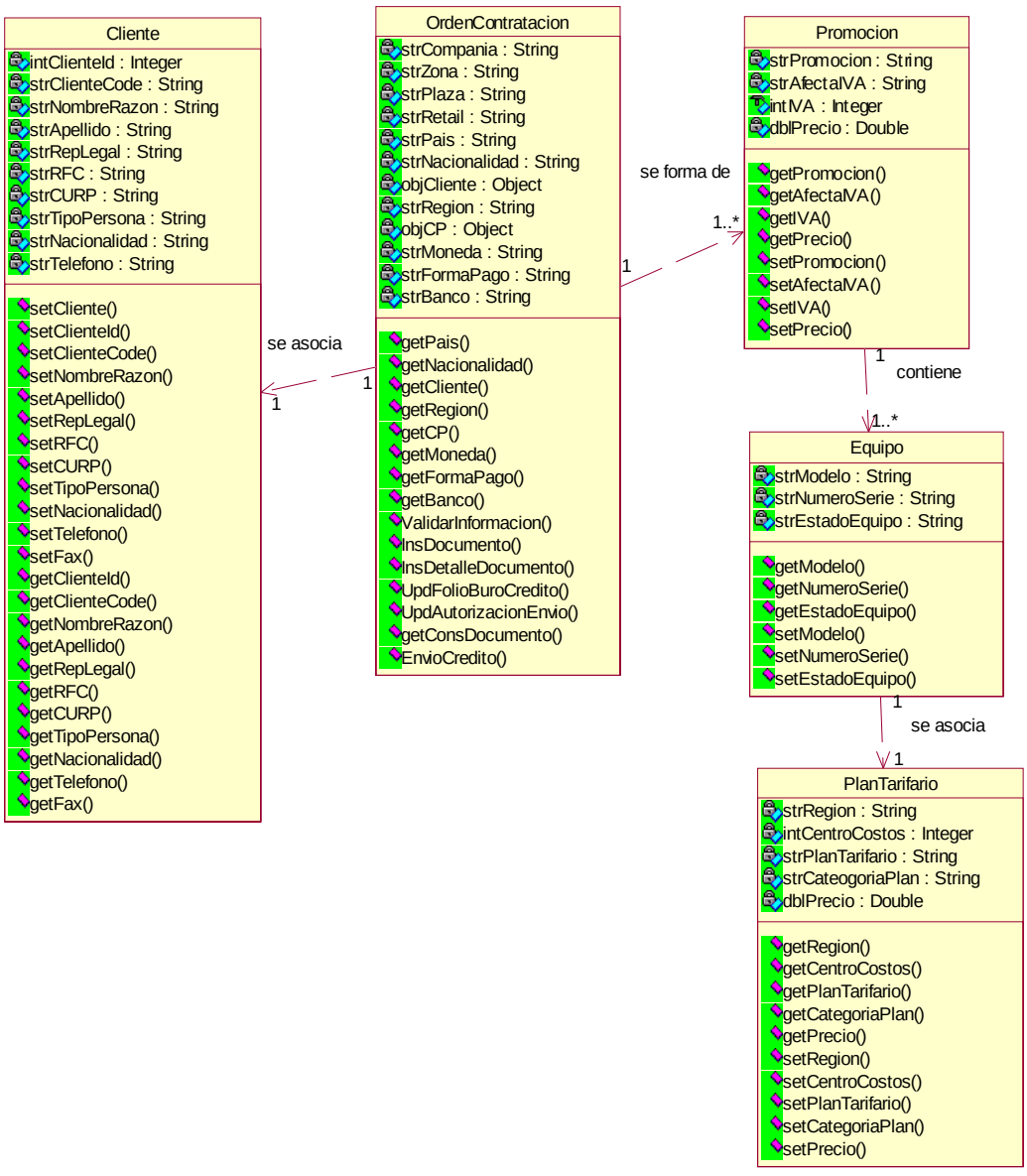
Diagrama de Clases y Diagramas de Objetos pertenecen a dos vistas complementarias del modelo. Un Diagrama de Clases muestra la abstracción de una parte del dominio. Un Diagrama de Objetos representa una situación concreta del dominio. Las clases abstractas no son instanciadas.

Generalización

Permite gestionar la complejidad mediante un ordenamiento taxonómico de clases, se obtiene usando los mecanismos de abstracción de Generalización y/o Especialización. La Generalización consiste en factorizar las propiedades comunes de un conjunto de clases en una clase más general. Los nombres usados: clase padre - clase hija. Otros nombres: superclase - subclase, clase base - clase derivada. Las subclases heredan propiedades de sus clases padre, es decir, atributos y operaciones (y asociaciones) de la clase padre están disponibles en sus clases hijas. La Generalización y Especialización son equivalentes en cuanto al resultado: la jerarquía y herencia establecidas. Generalización y Especialización no son operaciones reflexivas ni simétricas pero sí transitivas. La especialización es una técnica muy eficaz para la extensión y reutilización.
La noción de clase está próxima a la de conjunto. Dada una clase, podemos ver el conjunto relativo a las instancias que posee o bien relativo a las propiedades de la clase. Generalización y especialización expresan relaciones de inclusión entre conjuntos [CRANGEL].

Ejemplo de diagrama de clases

El siguiente fragmento del diagrama de clases muestra la relación entre las clases que se emplean para la generación de una orden de contratación en el punto de venta.



APÉNDICE H

Conceptos de base de datos relacionales

Entidades

Se define como entidad a cualquier objeto, real o abstracto, que existe en un contexto determinado o puede llegar a existir y del cual deseamos guardar información.

La entidades las podemos clasificar en:

Regulares	Aquellas que existen por sí mismas y que la existencia de un registro en la entidad no depende de la existencia de otro registro en otra entidad.
Débiles	Son aquellas entidades en las que se hace necesaria la existencia de otros registros de otras entidades distintas para que puedan existir registros en esta entidad.

Atributos

Las entidades se componen de atributos que son cada una de las propiedades o características que tienen las entidades. Cada registro de una misma entidad posee los mismos atributos, tanto en nombre como en número, diferenciándose cada uno de los registros por los valores que toman dichos atributos.

Existen cuatro tipos de atributos

Obligatorios	Aquellos que deben tomar un valor y no se permite ningún registro sin un valor determinado en el atributo.
Opcionales	Aquellos atributos que pueden tener valor o no tenerlo.
Llaves Primarias	Atributo que solo puede tener un único valor.
Llaves Foráneas	Aquellos atributos que pueden tener varios valores.

Existen atributos, llamados *derivados*, cuyo valor se obtiene a partir de los valores de otros atributos.

Tipos de relación

Se entiende por relación a la asociación, vinculación o correspondencia entre entidades.
Al igual que la entidades las relaciones se pueden clasificar en *regulares* y *débiles*, según estén asociando dos tipos de entidades regulares o una entidad débil con otra de cualquier tipo. Las relaciones débiles se subdividen en dos grupos:

En existencia	Cuando los registros de una entidad débil no pueden existir si desaparecen los registro de la entidad regular de la cual dependen.
En identificación	Cuando, además de ser una relación en existencia, los registros de una entidad débil no se pueden identificar por sí mismos y exigen añadir llave primaria de la entidad regular de la cual dependen para ser identificados. En este momento es cuando se dice que se hereda la llave primaria de la entidad regular a la entidad débil.

Cardinalidad

En cada relación se debe el número máximo y mínimo de registros de un tipo de entidad que pueden estar asociadas, mediante una determinada relación, con un registro de otra entidad. Este valor máximo y mínimo se conoce como cardinalidad y según corresponda se representa de la siguiente forma:

1 : n uno a muchos
n : 1 muchos a uno
n : n muchos a muchos

Reglas de normalización

La normalización es el proceso de organizar los datos en una base de datos. Esto incluye crear las tablas y establecer las relaciones entre ellas según las reglas diseñadas para proteger los datos y hacer la base de datos más flexible eliminando redundancia y la dependencia inconsistente.

Los datos redundantes desperdician espacio en disco y crean problemas de mantenimiento. Si deben cambiarse datos que existen en más de un lugar, los datos deben cambiarse exactamente de la misma manera en todas las tablas. Es más fácil implementar un cambio de dirección de cliente si ese dato sólo se guarda en la tabla de los clientes y en ninguna otra parte en la base de datos.

Qué significa dependencia inconsistente ? Mientras es intuitivo para un usuario buscar en la tabla de los Clientes la dirección de un cliente particular, no tiene sentido buscar allí el sueldo del empleado que llama a ese cliente. El sueldo del empleado se relaciona al empleado y por ello debe moverse a la tabla de empleados. Las dependencias inconsistentes pueden hacer difícil accesar a los datos porque el camino para encontrarlos puede ser desconocido.

Hay unas pocas reglas para la normalización de base de datos. Cada regla es llamada una *forma normal*. Si la primera regla se cumple, se dice que la base de datos está en *primera forma normal*. Si las primeras tres reglas se cumplen, se considera que la base de datos está en *tercer forma normal*. Aunque otros niveles de normalización son posibles, se considera que la tercera forma normal es suficiente para la mayoría de las aplicaciones.

Primera forma normal

- Elimine los grupos repetidos en las tablas individuales.
- Cree una tabla separada para cada conjunto de datos relacionados.
- Identifique cada conjunto de datos relacionados con una clave primaria.

No use varios campos en una sola tabla para guardar los datos similares. Por ejemplo, rastrear un artículo del inventario que puede venir de dos posibles fuentes, un registro del inventario podría contener los campos para Vendedor Código 1 y Vendedor Código 2.

Segunda forma normal

- Cree tablas separadas para los conjuntos de valores que se aplican a múltiples registros.
- Relacione estas tablas con una clave foránea.

Los registros deben depender exclusivamente de la clave primaria (una clave compuesta, si fuese necesario). Por ejemplo, considere la dirección de un cliente en un sistema de facturacion. La dirección se necesita en la tabla de los clientes, pero también en órdenes, envíos, facturas, etc. En lugar de guardar la dirección del cliente como una entrada separada en cada una de estas tablas, guárdelo en un lugar o en la tabla de los clientes o en una tabla de direcciones separada.

Tercera forma normal

- Elimine campos que no dependen de la clave primaria.

Los valores en un registro que no son parte de la clave de ese registro pueden no pertenecer a la tabla. En general, cuando un grupo de campos pueda aplicarse a más de un registro en la tabla, considere situar esos campos en una tabla separada.

EXCEPCIÓN: Adherir a la tercer forma normal, mientras es teóricamente deseable, no siempre es práctico. Si usted tiene una tabla de clientes y usted quiere eliminar todas las posibles dependencias, debe crear tablas separadas para las ciudades, códigos postal, representantes de ventas, tipos de clientes, y cualquier otro factor que pueda repetirse en varios registros. En la teoría, la normalización vale el esfuerzo. Sin embargo, muchas tablas pequeñas pueden degradar el funcionamiento o pueden exceder los archivos abiertos permitidos y las capacidades de memoria.

Puede ser más factible sólo aplicar la tercer forma normal a datos que frecuentemente cambian. Si algunos campos dependientes permanecen, diseñe su aplicación para exigirle al usuario que verifique todos los campos relacionados cuando cualquiera de ellos se modifique.

APÉNDICE I

Diccionario de datos del sistema

El diccionario de datos del sistema es el conjunto de objetos de base de datos que definen un esquema de base de datos; es decir, es el conjunto de tablas, relaciones, vistas, secuencias, *store procedures* (procedimientos almacenados), etc. En este apéndice me referiero solamente al diccionario de datos de las tablas del sistema.

La presentación del diccionario de datos se hará de acuerdo siguiente agrupamiento de información la cual se mencionó en capítulo 1:

- Roles o perfiles de trabajo
- Catálogos
- Inventario local
- Orden de contratación
- Cobranza de orden de contratación
- Contabilidad de operaciones
- Interfaces

Roles o pefiles de trabajo

Los roles o perfiles de trabajo permiten definir funciones o tareas específicas que un usuario de la aplicación puede realizar una vez que ha ingresado al sistema.

• ROLES	Almacena los roles del sistema.
• MODULOS	Almacena el árbol jerárquico de las opciones del menú del sistema.
• DETALLES_ROLES_MODULOS	Almacena la asociación de un rol con las opciones del menú.
• USUARIOS	Almacena la información de los usuarios del sistema.

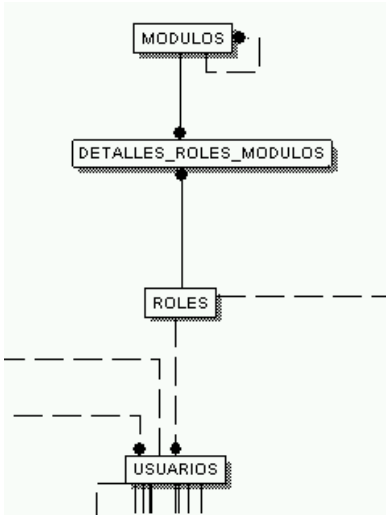


Figura I.1 Relación entre las tablas para perfiles de trabajo.

Catálogos

Los catálogos del sistema almacenan la información que la aplicación necesita para ejecutar diferentes procesos en diferentes etapas en el flujo de la información, de acuerdo a las reglas del negocio de la empresa. Los catálogos del sistema pueden dividirse en dos tipos; catálogos maestros y los catálogos maestro – detalle.

Catálogos Maestros

Los catálogos maestros son aquellos que constan de una sola entidad para almacenar información; es decir, no tienen dependencia con otras entidades para almacenar información inicial del sistema. A continuación menciono las tablas que pertenecen a esta clasificación:

- COMPANIAS — Almacena la base para la estructura organizacional de los puntos de venta.
- NACIONALIDADES — Almacena las nacionalidades para asociarlas al cliente en la generación de una orden de contratación.
- MONEDAS — Almacena las monedas para las operaciones del sistema.
- TIPOS_RECHAZOS — Almacena los tipos de rechazo para asociarlos a una orden de contratación en caso de no ser aprobada.
- PLANES_TARIFARIOS — Almacena la información de los precios de los planes tarifarios para las operaciones del sistema.
- FORMAS_PAGOS — Almacena las formas de pago para las operaciones del sistema.

- TIPOS_DISTRIBUCIONES — Almacena los tipos de distribución contable para las operaciones del sistema.
- BANCOS — Almacena los bancos para operaciones del sistema.
- PAQUETES — Almacena el encabezado de la información de los paquetes de equipos disponibles en el sistema.

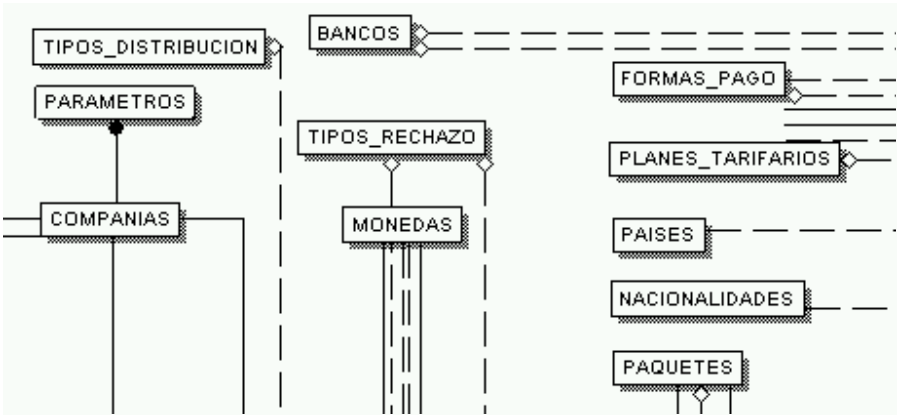


Figura I.2 Catálogos maestros del sistema

Catalogos Maestro – Detalle

Los catálogos maestro – detalle son aquellos en los que se hace necesaria la existencia de otros registros de otras entidades distintas para que puedan existir registros en esta entidad; es decir, tienen dependencia con otras entidades para almacenar información inicial del sistema, de tal forma que existe una entidad que funciona como encabezado y otra que funciona como detalle. A continuación menciono las tablas que pertenecen a esta clasificación:

- ZONAS — Almacena la información para el establecimiento de zonas y clasifcación de los puntos de venta.
- PLAZAS — Almacena la información de las plazas o ciudades donde se encuentran los puntos de venta.
- RETAILS — Almacena la información de los puntos de venta.
- ESTABLECIMIENTOS — Almacena la información del establecimiento comercial donde se ubica el punto de venta.
- DESTINOS MENSAJES — Almacena la información de las colas de mensajes donde será enviada la información de las órdenes de contratación por medio de Java Message Sservice (JMS) a las áreas de credito y activación para su evaluación y la activación de los equipos respectivamente.

- DETALLES_PAQUETES — Almacena la conformación de equipos que forma un paquete o promoción en el punto de venta.
- PRECIOS_PAQUETES — Almacena el precio de los paquetes del sistema.
- USUARIOS — Almacena la información y clasificación de los usuarios del sistema.
- CAJAS — Almacen la información de las cajas de los puntos de venta.
- TIPOS_VENTAS — Almacena los tipos de venta del sistema.
- TIPOS_SEGMENTOS — Almacena la información de los tipos de segmentos contables del sistema.
- TIPOS_CUENTAS — Almacena la información de los tipos de cuenta del sistema.
- CONFIGURACIONES_CONTABLES — Almacena la estructura de la configuración contable del sistema.
- CUENTAS_BANCARIAS — Almacena los número de las cuentas bancarias del sistema.
- PARAMETROS — Almacena la información de los parámetros del sistema.
- VENDEDORES — Almacena la información de los vendedores del punto de venta.

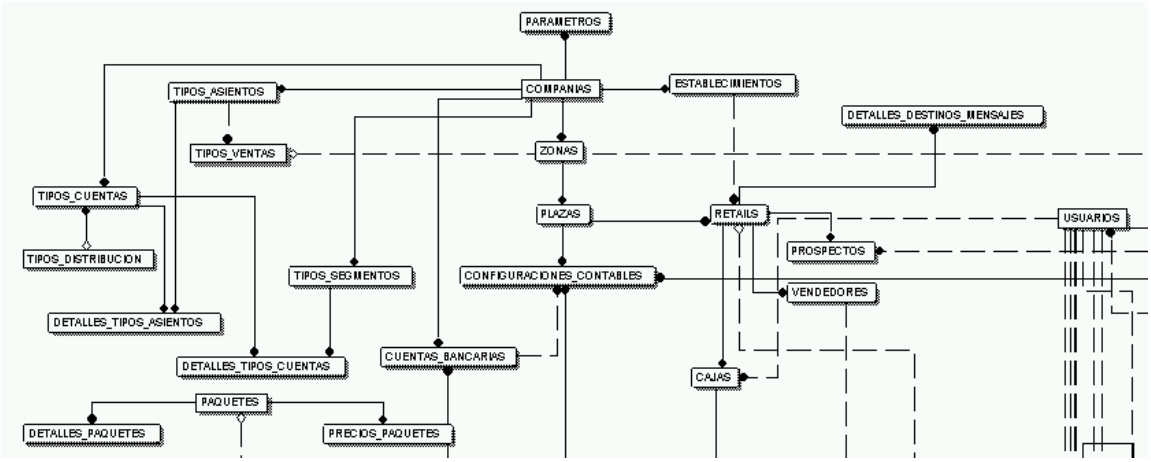


Figura I.3 Relaciones entre tablas maestro-detalle

Inventarios locales

La *Suite de Oracle Financial* es la aplicación que en uno de sus módulos concentra la información del inventario general de equipos. El término de inventario local se refiere a que por medio de una interfaz se pueda transferir la información de los equipos que desde la base de datos general de inventarios sean asignados al *Sistema de Comercialización en Línea* y que se almacene dicha información en la base de datos del sistema. El objetivo de este esquema es evitar hacer consultas a una base de datos de equipos muy grande y con esto disminuir el tiempo de respuesta en las transacciones de la operación diaria del sistema.

El inventario del sistema consta de una estructura maestro – detalle donde la tabla INVENTARIOS_LOCALES funciona como encabezado del inventario y la tabla DETALLES_INVENTARIOS_LOCALES almacena de forma detallada la información de los equipos que constituyen el inventario local de cada punto de venta. La clave de la compañía, zona, plaza o ciudad y retail o punto de venta permiten al sistema tener organizada por cada punto de venta la información de los equipos en los inventarios.

Esta sección utiliza una interfaz con el *ERP Oracle Financial Suite* en el módulo de *INVENTORY* para la recuperación de los números de serie de los equipos que son asignados a un punto de venta.

Consulte la sección de *interfaces* de este capítulo para obtener más información del *ERP Oracle Financial Suite*.

Las tablas involucradas en este proceso son las siguientes:

- RETAILS — Almacena la información de los puntos de venta.
- USUARIOS — Almacena la información y clasificación de los usuarios del sistema.
- INVENTARIOS_LOCALES — Almacena la información del encabezado del inventario local del punto de venta.
- DETALLES_INVENTARIOS_LOCALES — Almacena la información de los equipos y su estado en el inventario local del punto de venta.

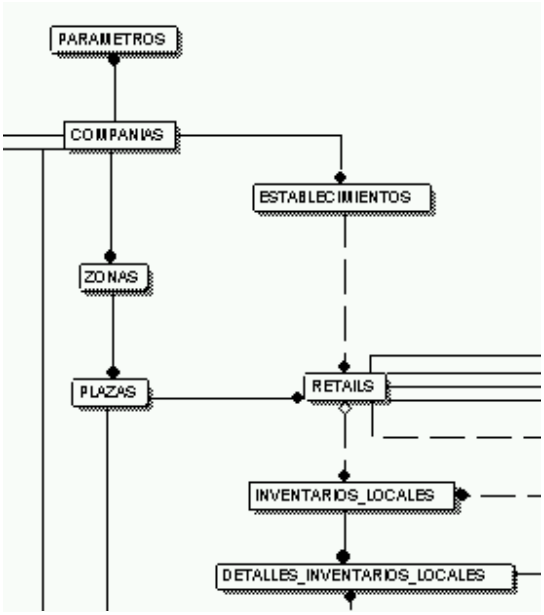


Figura I.4 Relación entre tablas del inventario local

Orden de contratación

La estructura que almacena la información de las órdenes de contratación tiene la forma de un maestro – detalle, de tal manera que la tabla DOCUMENTOS funciona como encabezado de la información de la orden de contratación, la tabla DETALLES_DOCUMENTOS almacena la información detallada de los equipos que son asociados a una orden de contratación y la tabla DETALLES_DOCUMENTOS_SERVICIOS almacena la información de los servicios que son asociados a cada uno de los equipos que integran una orden de contratación.

Las tablas involucradas en este proceso son las siguientes:

- BANCOS — Almacena los bancos para operaciones del sistema.
- MONEDAS — Almacena las monedas para las operaciones del sistema.
- FORMAS_PAGOS — Almacena las formas de pago para las operaciones del sistema.
- TIPOS_RECHAZOS — Almacena los tipos de rechazo para asociarlos a una orden de contratación en caso de no ser aprobada.
- REGIONES — Almacena la información de las regiones para las operaciones del sistema.
- USUARIOS — Almacena la información y clasificación de los usuarios del sistema.

- RETAILS — Almacena la información de los puntos de venta.
- PAQUETES — Almacena el encabezado de la información de los paquetes de equipos disponibles en el sistema.
- PLANES_TARIFARIOS — Almacena la información de los precios de los planes tarifarios para las operaciones del sistema.
- PROSPECTOS — Almacena la información de un posible cliente cuya orden de contratación fue rechazada.
- PAISES — Almacena la información de los paises para las operaciones del sistema.
- DOCUMENTOS — Almacena la información del encabezado de la orden de contratación.
- DETALLES_DOCUMENTOS — Almacena la información de los equipos que son asociados a una orden de contratación.
- DETALLES_DOCUMENTOS_SERVICIOS — Almacena la información de los servicios que son asociados a los equipos de una orden de contratación.

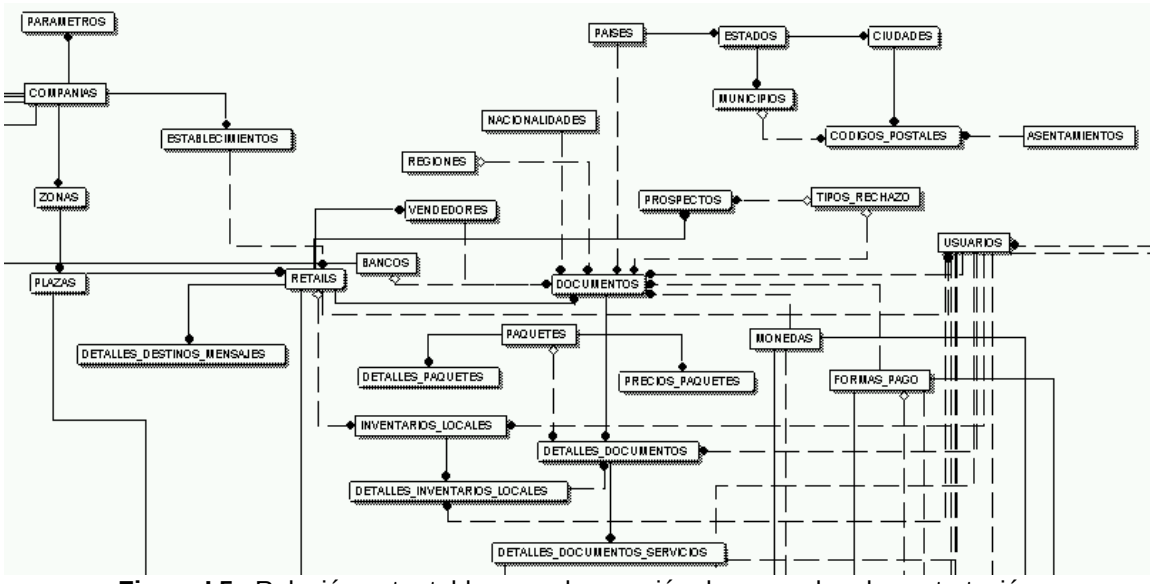


Figura I.5 Relación entre tablas para la creación de una orden de contartación

Cobro de orden de contratación

La tabla COBRANZAS funciona como la intersección entre algunos catálogos y las tablas donde se almacena la información de las órdenes de contratación, esta tabla contiene la estructura que almacena el detalle de la cobranza de una orden de contratación.

Esta sección utiliza una interfaz con el sistema *BSCS* (Business Support and Control System*)* para la obtención del número de cuenta de un cliente nuevo.

Consulte la sección de *interfaces* de este capítulo para obtener más información del sistema *BSCS*.

Las tablas involucradas en este proceso son las siguientes:

- RETAILS — Almacena la información de los puntos de venta.
- DOCUMENTOS — Almacena la información del encabezado de la orden de contratación.
- MONEDAS — Almacena las monedas para las operaciones del sistema.
- FORMAS_PAGOS — Almacena las formas de pago para las operaciones del sistema.
- BANCOS — Almacena los bancos para operaciones del sistema.
- CORTES_CAJAS — Almacena la información de los cortes de caja de un punto de venta.
- TIPOS_VENTAS — Almacena los tipos de venta para las operaciones del sistema.
- USUARIOS — Almacena la información y clasificación de los usuarios del sistema.
- COBRANZAS — Almacena la información del detalle del cobro de una orden de contratación.

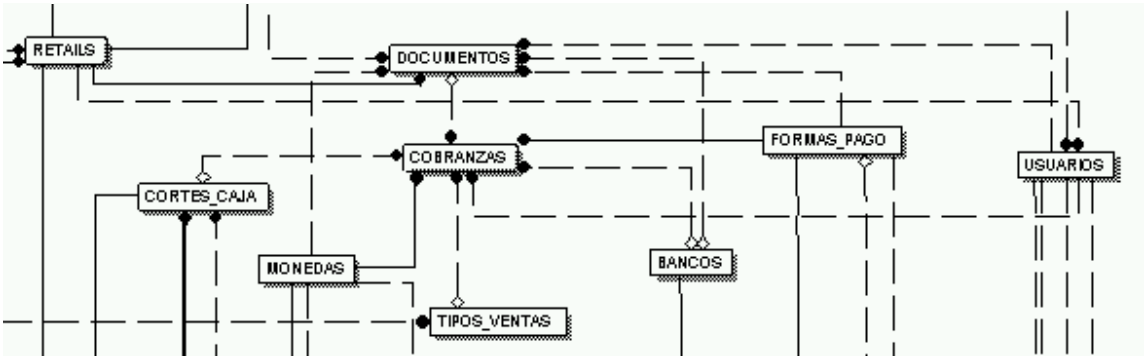


Figura I.6 Relaciones entre tablas para el cobro de una orden de contratación

Contabilidad de operaciones

La estructura que almacena la información contable de las ventas del día tiene la forma de varios maestro – detalle, en esta estructura se almacena la contabilidad de la operaciones que cada punto de venta ha realizado desde el último corte de caja.

Esta sección utiliza una interfaz con el *ERP Oracle Financial Suite* en el módulo de *GENERAL LAYER* para la transferencia de las pólizas contables que se generan en un corte de caja. Así mismo utiliza interfaces con los módulos *ACCOUNT RECEIVABLES* e *INVENTORY* para la transferencia de la información de las facturas emitidas y la transferencia de los números de serie de los equipos que estan asociados a éstas respectivamente.

Consulte la sección de *interfaces* de este capítulo para obtener más información del *ERP Oracle Financial Suite*.

Las tablas involucradas en este proceso son las siguientes:

- RETAILS — Almacena la información de los puntos de venta.
- USUARIOS — Almacena la información y clasificación de los usuarios del sistema.
- MONEDAS — Almacena las monedas para las operaciones del sistema.
- FORMAS_PAGOS — Almacena las formas de pago para las operaciones del sistema.
- TIPOS_CUENTAS — Almacena la información de los tipos de cuentas contables para las operaciones del sistema.
- TIPOS_SEGMENTOS — Almacena la información de los tipos de segmentos contables para las operaciones del sistema.
- TIPOS_ASIENTOS — Almacena los tipos de asientos contables para las operaciones del sistema.
- CAJAS — Almacen la información de las cajas de los puntos de venta.
- CORTES_CAJAS — Almacena la información de los cortes de caja de un punto de venta.
- DETALLES_CORTES_CAJAS — Almacena los detalles de los cortes de caja de los puntos de venta.
- POLIZAS — Almacena el encabezado de las pólizas contables del sistema.
- DETALLES_POLIZAS — Almacena el detalle de las pólizas contables del sistema.
- DETALLES_TIPOS_CUENTAS — Almacen el detalle de los tipos de cuentas contables para las operaciones del sistema.
- DETALLES_TIPOS_ASIENTOS — Almacena el detalle de los tipos de asientos contables para las operaciones del sistema.

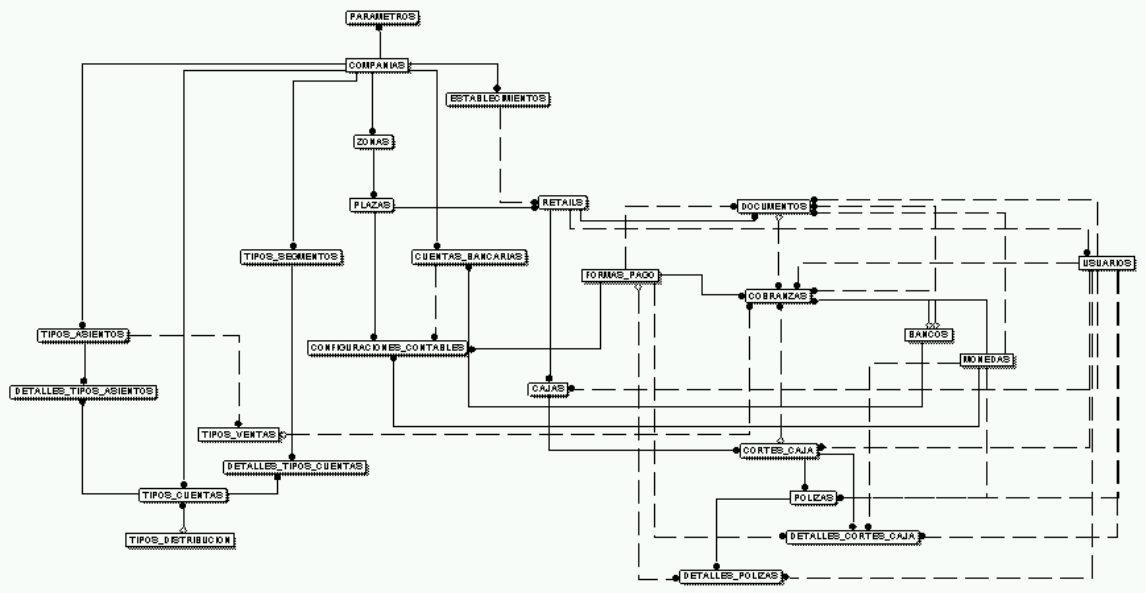


Figura I.7 Relaciones entre tablas para la contabilidad de operaciones

Interfaces

Las interfaces con que cuenta el sistema le permiten a la aplicación interactuar con otros sistemas de información existentes en la empresa y mantener la integridad de los datos entre todos los sistemas involucrados. Los siguientes sistemas son los que mantienen relación con el *Sistema de Comercialización en Línea*:

BSCS

Por sus siglas en ingles BSCS significa *Business Support and Control System*, y es un conjunto de aplicaciones altamente flexible que pueden ser fácilmente modificadas para satisfacer cada entorno específico de negocios del cliente. Los módulos incluyen un motor de tasación que procesa las llamadas transmitidas de los centros de alternación móvil. BSCS es el mejor sistema de facturación celular utilizado en la industrial de las telecomunicaciones.

La interfaz desarrollada hacia este sistema permite recuperar la secuencia numérica para la formación del número de cliente que se presentará en la factura de equipo y en el comprobante de pago. Además utiliza una interfaz adicional para enviar para enviar al sistema de BSCS la información personal del cliente y de los equipos vendidos, la cual será utilizada en los procesos de facturación mensuales de la empresa.

ERP Oracle Financial Suite

Oracle Financial es parte de Oracle Applications, una suite integrada de más de 45 módulos para admistración financiera, administración de cadenas de abastecimiento, manufactura, recursos humanos y ventas que permite automatizar los procesos de las empresas.

¿ Qué es un ERP ?

Por sus sigla en ingles ERP (*Enterprise Resource Planning*) es un sistema muy sofisticado que reduce la presión y la carga de trabajo de los administradores y proporciona información exacta y oportuna para tomar apropiadas decisiones de negocio.

Componentes de un Oracle ERP

Para el manejo fácil de los sistemas de administración de negocios el ERP ha sido dividido en los siguientes módulos:

1. Cuentas a pagar / a cobrar
2. Administración de activos
3. Facturación de materiales
4. Planeación de capacidad de solicitudes
5. Comercio electrónico
6. Contabilidad financiera
7. Contabilidad
8. Administración de recursos humanos
9. Logística
10. Calendarización
11. Planeación de solicitudes de material
12. Nómina
13. Compras y recursos
14. Ventas y mercadotecnia

Los subsistemas con los cuales se desarrollaron interfaces son los siguientes:

- **Contabilidad**

  Permite la administración total de la información, es una solucion integral en la administración financiera que mejora considerablemente el control de las finanzas, la recopilación de datos, el acceso a la información y el reporteo financiero de la empresa.

  La interfaz desarrollada hacia este módulo permite enviar la información de las ventas realizadas en los puntos de venta para su contabilización en el módulo de contabilidad.

- **Cuentas a cobrar**

  Es un módulo con una completa funcionalidad en cuentas a cobrar que que mejora el flujo y manejo de efectivo de las necesidades de facturación.

  La interfaz desarrollada hacia este módulo permite enviar la información de las facturas emitidas así como la información de los pagos recibidos por concepto de las ventas de equipo en los puntos de venta.

- **Inventarios**

  Es un módulo con una completa funcionalidad para la administración de inventarios de la cadena fuente, incrementa la eficiencia operacional a través del mejoramiento del movimiento de material mientras que proporciona un estricto control del mismo.

  La interfaz desarrollada hacia este módulo permite enviar la información de los equipos vendidos desde el último cierre de operaciones para que sea actualizada la información de la base de datos general de inventarios de la empresa.

Diagrama entidad relación

El diagrama Entidad / Relación (E/R) es la representación gráfica de las relaciones entre las entidades que conforman un sistema.

APÉNDICE J

Componentes J2EE

Las aplicaciones J2EE son hechas de componentes. Un componente J2EE es una unidad de software funcional contenida en sí misma que es integrada en una aplicación J2EE con sus clases relacionadas y archivos que están comunicados con otros componentes. La especificación J2EE define los siguientes componentes J2EE:

- Aplicaciones cliente y applets son componentes que se ejecutan en la máquina del cliente.
- La tecnología Java Servlet y JavaServer Page™ (JSP™) son componentes web que se ejecutan en el servidor.
- Los componentes Enterprise JavaBeans™ (EJB™) son componentes de negocio que se ejecutan en el servidor.

Los componentes J2EE son escritos en el lenguaje de programación Java y son compilados de la misma forma como cualquier programa en un lenguaje. La diferencia entre los componentes J2EE y las clases Java estándar es que los componentes son integrados dentro de una aplicación J2EE, se verifica que esté bien formada y en cumplimiento con la especificación J2EE, y luego es publicada en producción, donde ellas son ejecutadas y administradas por el servidor J2EE [SUN].

Solamente se explicarán los componentes Java Servlet y JavaServer Page™ y los componentes Enterprise JavaBeans™, debido a que fueron los componentes empleados en el desarrollo del sistema.

Componentes web

Los componentes J2EE web pueden ser igual servlets o páginas JSP. Los servlets son clases en lenguaje de programación Java que dinámicamente procesan peticiones y construyen respuestas. Las páginas JSP son documentos basados en texto se ejecutan como servlets pero permiten aproximación más natural para crear contenido dinámico.

Las páginas HTML y los applets con asociados con los componentes web durante la construcción de la aplicación, pero no son considerados componentes web por la especificación J2EE. El servidor utiliza clases que también son asociadas con componentes web y, como las páginas HTML, no son consideradas componentes web.

Componentes de negocio

El código de negocio, el cual es lógico que resuelve o reune las necesidades del dominio de un negocio en particular es operado por los *enterprise beans* que se ejecutan en la capa de negocio. La figurá J.1 muestra como un *enterprise beans* recibe datos de programas del cliente, los procesa (si es necesario), y los envia a la capa de *sistemas de información empresarial* (EIS) para almacenarlos. Un *enterprise beans* también puede regresar datos almacenados, procesarlos (si es necesario), y enviarlos de regreso al programa del cliente.

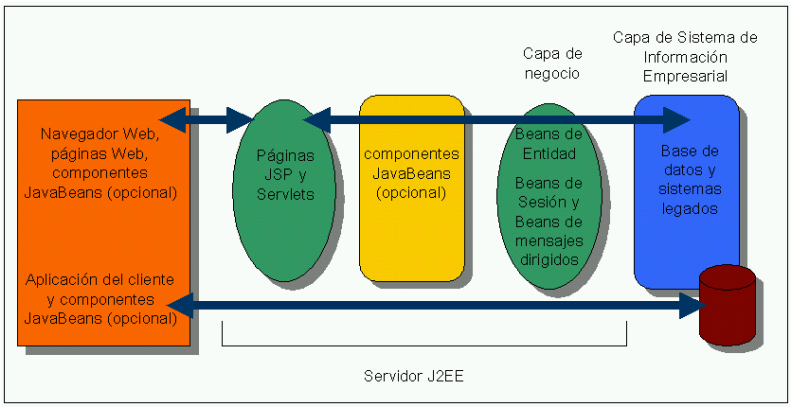


Figura J.1 Capa de Negocios y EIS

Existen tres tipos de *enterprise beans*: *session beans*, *entity beans*, y *message-driven beans*. Un *session beans* representa una comunicación temporal con un cliente. Cuando el cliente finaliza la ejecución, el *session beans* y sus datos desaparecen. Por el contrario, un *entity beans* representa un dato persistente almacenado en un registro de una tabla de base de datos. Si el cliente termina o si el servidor es dado de baja, el servicio asegura que el dato del *entity bean* sea grabado.

message-driven bean combina características de un *session bean* y un escuchador de mensajes Java Message Service (JMS), permitiendo a un componente de negocio recibir mensajes JMS asíncronamente [SUN].

Capa de sistema de información empresarial (EIS)

La capa de sistema de información empresarial maneja software del sistema de información empresarial e incluye sistemas de infraestructura empresarial tales como planeación de recursos empresariales (*Enterprise Resource Planning*, ERP), procesamiento de transacciones mainframe, sistemas de base de datos, y otros sistemas de información legados. Los componentes de aplicaciones J2EE pueden necesitar accesar a sistemas de información empresarial para conectvidad de bases de datos, por ejemplo [SUN].

APÉNDICE K

Servlets

Un servlet es una tecnología Java basado en componentes web, administrado por un contenedor, que genera contenido dinámico. Como otros componentes basados en Java, los servlets son plataformas independientes de las clases Java que son compiladas a una plataforma neutral bytecode que pueden ser cargadas adentro dinámicamente y ejecutarse por un servidor web Java habilitado. Los contenedores (*containers*) algunas veces son llamados *servlet engines*, estos son extensiones de servidores web que proporcionan funcionalidad a los servlets. Los servlets interactuan con un cliente web vía una petición / respuesta (request / response) implementada por el contenedor de servlets.

Contenedor de Servlets (*Servlet Container*)

El contenedor de servlets es una parte del web Server o Servidor de Aplicaciones (*Application Server*) que proporciona los servicios de red sobre los cuales las peticiones y respuestas son enviadas.

Un contenedor de servlets puede estar construido dentro del host de un web Server, o instalado como un componente adicional a un web Server via un API o extensión nativa del servidor.

Todos los contenedores de servlets deben soportar el protocolo http para peticiones y respuestas, pero adicionalmente las peticiones / respuestas basadas en protocolos HTTPS (HTTP sobre SSL) pueden ser soportados. La versión mínima requerida de la especificación HTTP que un contenedor puede implementar es HTTP/1.0. Es ampliamente sugerido que los contenedores implementen mejor la especificación HTTP/1.1.

Comparación de servlets con otras tecnologías

Los servlets tienen las siguientes ventajas sobre otras tecnologías

- Son generalmente mucho más rápidos que los scripts CGI (Comun Gateway Interface) porque un modelo diferente de proceso es utilizado.
- Utilizan un API (Application Program Interface) estándar que es soportado por la mayoría de los web Server.
- Tienen todas las ventajas del lenguaje de programación Java, incluyendo el facilitar el desarrollo y la independencia de la plataforma.
- Pueden accesar a un amplio conjunto de API's disponibles para la plataforma Java.

La interfaz del servlet

La interfaz del servlet es la abstracción central del API del servlet. Todos los servlets utilizan esta interfaz directamente, o más comúnmente, por la extensión de una clase que utilice la interfaz. Las dos clases en el API del servlet que utilizan la *interfaz del servlet* son GenericServlet y HttpServlet.

Métodos de manejo de peticiones (Request Handling Methods)

La interfaz básica del servlet define un método *service* para el manejo de peticiones del cliente. Este método es llamado por cada petición que el contenedor de servlets envía a una instancia del servlet.

El manejo de peticiones concurrentes al *web Application* generalmente requiere que el desarrollador web diseñe servlets que puedan trabajar con múltiples hilos que se ejecuten dentro del método *service* en un momento en particular.

Generalmente el *web Container* maneja las peticiones concurrentes con el mismo servlet para ejecuciones concurrentes del método *service* en diferentes hilos.

Métodos de manejo de peticiones específicas de HTTP (HTTP Specific Request Handling Methods)

La subclase abstracta HttpServlet agrega métodos adicionales más alla de la interfaz básica del servlet los cuales son automáticamente llamados por el método *service* en la clase HttpServlet para ayudar en el procesamiento de peticiones basadas en HTTP. Los métodos son:

- doGet para menejo de HTTP GET Request
- doPost para menejo de HTTP POST Request
- doDelete para menejo de HTTP DELETE Request
- doHead para menejo de HTTP HEAD Request
- doOptions para menejo de HTTP OPTIONS Request
- doTrace para menejo de HTTP TRACE Request

Generalmente cuando el desarrollo de Servlets esta basado en HTTP, el desarrollador de Servlets se enfocará en los métodos doGet y DoPost.

Ciclo de vida del Servlet

Un Servlet es manejado a través de un bien definido ciclo de vida que define como es cargado, instanciado e inicializado, el manejo de peticiones de clientes, y como es puesto fuera de servicio. Este ciclo de vida es manejado en el API por los métodos, *init*, *service* y *destroy* de la interfaz javax.servlet.Servlet que todos los Servlets deben de manejar directamente, o indirectamente a través de las clases abstractas GenericServlet o HttpServlet [SUN].

Funcionalidad del Servlet

- **Leer los datos enviados por el usuario.**
  Por lo regular estos datos son introducidos en un formulario o en una página web, pero también podrían provenir de algún subprograma (applet) Java.
- **Buscar cualquier otra información respecto a la petición que esté incrustada en la petición HTTP.**
  Esta información incluye los detalles respecto a las facultades del navegador, las cookies, el nombre del equipo host del cliente que hace la petición.
- **Generar resultados.**
  Este proceso podría necesitar comunicarse con una base de datos, ejecutar una llamada RMI o CORBA, invocar una aplicación existente o calcular la respuesta directamente.
- **Dar formato a los resultados dentro de un documento.**
  En la mayoría de los casos, esto trae consigo insertar información en una página HTML.
- **Establecer los parámetros http de respuesta adecuados.**
  Esto se traduce en decirle al navegador el tipo de documento que será devuelto (por ejemplo, HTML), establecer las cookies y ocultar los parámetros, etc.
- **Devolver el documento al cliente.**
  Este documento podría ser enviado en un formato de texto (HTML), binario (imágenes GIF), o incluso en un formato comprimido como gzip [HALL01].

Ejemplo de código fuente de Servlets

A continuación se muestra como ejemplo un fragmento de código de un servlet que fue desarrollado en el *sistema de comercialización en línea* para validar los servidores disponibles de JMS para el envio de mensajes.

Ejemplo de implementación de un Servlet

Nombre del Archivo: JmsCredito
Propósito: Revisa los servidores disponibles de JMS para el envio de mensajes.

```
import java.io.*;
import java.util.*;
import java.text.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.rmi.RemoteException;
import java.net.URLEncoder;
import javax.naming.*;
import javax.rmi.*;
import javax.jms.*;
import classes.*;
import OrdenContratacionBeans.*;
import controlador;

public class JmsCredito extends controlador
{

   public void revisaServidorJMS(javax.servlet.http.HttpSession session,
                                 javax.servlet.http.HttpServletRequest request) {
        OrdenCont   ordCont;
        Vector          vdatos     = new Vector();
        Vector          vJMSsrv = new Vector();
        String           vsrv        = new String();

        try{
                session                     = request.getSession();
                vJMSsrv                     = this.getQueues(this.getQueue());
                OrdenContratacion  vordTemp = null;
                vsrv                        = null;
                // Barrer los servidores disponibles para credito
                for (int i=0 ; i<vJMSsrv.size() ; i++){
                    System.out.println("("+ this.vuser +")Verificando en servidor: " +
                                                    vJMSsrv.get(i));
                    //vordTemp = this.getMensaje((String)vJMSsrv.get(i),this.vuser,request);
                    if (vordTemp != null) {
                        //vsrv = (String)vJMSsrv.get(i);
                        System.out.println("vsrv:"+vsrv);
                        break;
                    }
                }
                if (vordTemp != null){
                    this.vcompania = vordTemp.getCompania();
                    this.vzona          = vordTemp.getZona();
                    this.vplaza         = vordTemp.getPlaza();
                    this.vretail   = vordTemp.getRetail();

                    //verificamos si la orden existe y/o ya fue evaluda
                    //if ( isEvaluada(vcompania,vzona,vplaza,vretail,vuser,vextra,
                            vordTemp.getNumDocumento()) == 1 ){
                    if ( this.isEvaluada(vordTemp.getNumDocumento()) == 1 ) {
```

```
                        System.out.println("Evaluar Orden..." + vordTemp.getNumDocumento());

                        // Regresa mensaje a la cola...
                        this.RegresaMensaje(vsrv,this.vuser,vordTemp);
                        //subimos el objeto a la session para poder cargarlo mas adelante
                        session.setAttribute("OrdenPorEvaluar", vordTemp);
                        this.setSubmit(true);}
                    else{
                            System.out.println("Orden \""+ vordTemp.getNumDocumento() + "\"
                              ya fue evaluda, se baja de Queue");
                            setRefresh(true);
                            this.setRefreshTime(5);}}
                else{
                        System.out.println("No se recibio ningun mensaje en Credito");
                        this.setRefresh(true);
                        this.setRefreshTime(60);}
        }
        catch (Exception e) {
                e.printStackTrace ();
                //    caso ="<br>" + e.getMessage();
                System.out.println(e.getMessage());
                this.Reset();
        }
   }
}
```

APÉNDICE M

JavaServer Pages

JavaServer Pages es una tecnología de la plataforma Java 2 Enterprise Edition (J2EE), para construir aplicaciones para contenido web dinámicamente generado, tal como HTML, DHTML, XHTML y XML. La tecnología JavaServer Pages permite la fácil creación de páginas web que crean contenido dinámico de gran potencia y flexibilidad.

Beneficios de la tecnología JavaServer Pages

La tecnología JavaServer Pages ofrece los siguientes beneficios:

- **Escríbalo una vez, ejecútelo donde sea**

  La tecnología JSP es una plataforma independiente en sus páginas web dinámicas, en su servidores web, y en sus componentes del servidor fundamentales. Las páginas JSP pueden ser escritas en cualquier plataforma, ejecutarse en cualquier servidor web o servidor de aplicaciones permitido e ingresado por cualquier navegador web. Los componentes del servidor pueden ser construidos sobre cualquier plataforma y ejecutados en cualquier servidor.

- **Soporte en herramientas de alta calidad**

  La independencia de la plataforma permite a los usuarios de JSP escoger el mejor tipo de herramienta. Adicionalmente, el objetivo explicito del diseño de páginas JSP es permitir la creación de herramientas portables de alta calidad.

- **Reuso de componentes y librerías tag**

  La tecnología JavaServer Pages resalta la reutilización de compomentes, como por ejemplo los componentes JavaBeans, Enterprise Java Beans y librerías tag. Esos componentes pueden ser utilizados con herramientas interactivas para el desarrollo de componentes y composición de páginas, reduciendo considerablemente el tiempo dedicado al desarrollo. En resumen, ellos proporcionan el potencial y la flexibilidad del lenguaje de programación Java.

- **Separación del contenido dinámico y estático**

  La tecnología JavaServer Page permite la separación del contenido estático en una plantilla del contenido dinámico que es insertado en una plantilla estática. Esto simplifica de manera significativa la creación de contenido. La separación es soportada por componentes diseñados espacialmente para la interacción con objetos del lado del servidor y por mecanismos de extensión tag.

- **Soporte para acciones, expresiones y scripts**

  La tecnología JavaServer Page soporta la escritura de elementos así como también de acciones. Las *acciones* de encapsulamiento tienen un funcionalidad muy útil en una forma conveniente que puede ser manipulada por herramientas. Las *expresiones* son utlilizadas para accesar a datos. Los *scripts* pueden ser utilizados para conjuntar esta funcionalidad en la forma de una página.

- **Arquitectura de aplicaciones empresariales de acceso a web**

  La tecnología JavaServer Page es una parte integral de la plataforma Java 2 Enterprise Edition (J2EE). Ahora se pueden desarrollar poderosas aplicaciones de capa intermedia que incluyan un *web site* que utilize la tecnología JavaServer Pages como una interfaz de usuario para componentes Enterprise Java Beans en un cómodo ambiente J2EE [SUNJSP].

Ejemplo de código fuente de JavaServer Pages

A continuación muestro uno de los JavaServer Page que se realizaron en el Sistema de Comercialización en Línea:

Nombre del Archivo: login.jsp
Propósito: Validar la clave de usuario y contraseña del usuario, así como recuperar diferentes atributos cuando el usuario ha sido validado en el sistema.

```
<%@ page language="Java" import="CDBManager
  , java.util.*
  , javax.rmi.PortableRemoteObject
  , javax.naming.InitialContext
  , java.sql.*
  , javax.sql.*
  , Cifrar" contentType="text/html;charset=WINDOWS-1252"
%>

<%!
    String vusuario   = "";
    String vcontrasena= "";
    String vquery     = "";
    String vauxiliar  = "";
    String dbName     = "jdbc/RetailInit";
    CDBManager bd;
    Cifrar llave      = new Cifrar();
%>
<%
    try{
         vusuario     =      request.getParameter("vlogin");
         vcontrasena  =      request.getParameter("vpass");
         if(vusuario==null || vusuario.length()<1 ||
            vcontrasena==null || vcontrasena.length()<1 ){
%>
<%@ include  file="../html/login.html" %>
<%       }
         else {
            InitialContext ictx  = new InitialContext();
            DataSource dsrc = (DataSource) ictx.lookup(dbName);
            Connection con = null;
            //si hay una conexion abierta la invalidados
            if(!(session.isNew())){
                try{
                   con = (Connection)session.getAttribute("conexion") ;
                   //si existe la conexion revisamos si esta abierta (proceder a cerrrarla
                   //y despues la eliminamos de la session
                   if(!(con.isClosed()))
                      con.close();
                   session.removeAttribute("conexion") ;
                   session.removeAttribute("usu_clave") ;
                   session.removeAttribute("usu_pass") ;
                  }catch(NullPointerException npe1){
                    //si entro aqui es porque no existe el atributo en la session
                    //out.println("aqui"+ npe1.getMessage());
         }
       }
       //Iniciar conexion bd
       try{
             //creamos la nueva conexion
             con =  dsrc.getConnection(vusuario,vcontrasena);
             //inicia una nueva session
```

```
                session = request.getSession(true);
                //agrega a la session la el objeto de coneccion
                session.setAttribute("conexion", con);
                //agrega a la session el nombre del usuario
                session.setAttribute("usu_clave", vusuario.toUpperCase());
                //cargamos el password cifrado
                session.setAttribute("usu_pass", llave.codificar(vcontrasena.toUpperCase()) );
                //cargamos parametros adicionales zona,plaza, zona,retail
                vquery = "SELECT USU_RET_PLA_ZON_COM_CLAVE " +
                        ",USU_RET_PLA_ZON_CLAVE " +
                        ",USU_RET_PLA_CLAVE " +
                        ",USU_RET_CLAVE " +
                        ",USU_ROL_CLAVE " +
                        " USU_APLICACION " +
                        " FROM USUARIOS " +
                        " WHERE USU_CLAVE = '"+ VUSUARIO.TOUPPERCASE() +"' "+
                        " AND USU_STATUS='A'";
                bd      = new CDBManager(con);
                System.out.println(vquery);
                bd.loadResult(vquery);
                if(bd.siguiente()){
                     //agrega a la sesión el tipo de apliacion
                     session.setAttribute("usu_aplicacion",bd.getStr("usu_aplicacion"));
                     //agrega a la session el rol
                     session.setAttribute("rol_clave",bd.getStr("usu_rol_clave"));
                     //revisamos el rol del usuario
                     //si este es un usuario que puede trabajar en un retail difrente al
                     //asignado en la base de datos, vgr: Administrador o gelp desk
                     //mandamos la pantalla para que seleccione el retail
                     //de lo contrario almacenamos el resto de los valores a la session.
                     //los posibles valores de aquellos roles que tenga permitido trabajar en
                     //diferentes retails sin realizar el cambio a nivel de bd derán de
                     especificarse
                     //al evaluar el valor de usu_rol_clave en este jsp, como se realiza a
                     continuación
                     vauxiliar = bd.getStr("usu_rol_clave").toLowerCase();
                     if(vauxiliar.indexOf("adm")!=-1)
                     {
                         bd.cierraStatement();
                        //realiza la llamada la llamada al jsp que realizara la firma del
                        usuario
%>
                         <jsp:forward page="../jsp/setParameter.jsp"/>
<%
                         return;
                     }
                     //agrega a la session la compañía
                     session.setAttribute("com_clave",
                     bd.getStr("usu_ret_pla_zon_com_clave"));
                     //agrega a la session la zona
                     session.setAttribute("zon_clave", bd.getStr("usu_ret_pla_zon_clave"));
                     //agrega a la session la plaza
                     session.setAttribute("pla_clave", bd.getStr("usu_ret_pla_clave"));
                     //agrega a la session el retail
                     session.setAttribute("ret_clave", bd.getStr("usu_ret_clave"));
                     //despliega pantalla principal del sistema
%>
                     <%@ include file="../html/main.html" %>
<%
                 }
                 else
                 {
                     //el usuario esta inhabilitado
                     out.println("<html><head><title>Error de Acceso al Sistema
                     Retail</title>");
                     out.println("<link rel='stylesheet' type='text/css'
                     ref='../webapp/css/work.css'>");
                     out.println("</head><body bgcolor=white>");
                     out.println("<form name=frm method=get>");
                     out.println("<input type=hidden>");
                     out.println("<br><br><br>");
```

```
                    out.println("<div class=Error align=center>");
                    out.print("<img src='../webapp/images/Nextel.gif' border=0>");
                    out.println("<br>");
                    out.println(" Lo sentimos, su usuario ha sido bloqueado.");
                    out.println("<br>");
                   out.println(" Comuniquese con su administrador para
                    cualquier duda o aclaración.");
                    out.println("</div>");
                    out.println("</form>");
                    out.println("</body></html>");
                }
            }
                catch(Exception e)
        {
                    out.println("<html><head><title>Error de Acceso al Sistema
                    Retail</title>");
                    out.println("<link rel='stylesheet' type='text/css'
                    href='../webapp/css/work.css'>");
                    out.println("</head><body bgcolor=white>");
                    out.println("<form name=frm>");
                    out.println("<input type=hidden>");
                    out.println("<br><br><br>");
                    out.println("<div class=Error align=center>");
                    out.print("<a href='../html/login.html'>");
                    out.print("<img src='../webapp/images/Nextel.gif' border=0></a>");
                    out.println("<br>");

                    if((e.getMessage()).startsWith("ORA-01017"))
                        out.println(" El usuario y/o contrase&ntilde;a proporcionados no son
                        validos.");
                    else
                       if((e.getMessage()).startsWith("ORA-01045"))
                          out.println("No tiene permiso para iniciar un sesión en la base de
                          datos.");
                      else
                          out.println("Error al iniciar su sesión:<br>"+e.getMessage());
                          out.println("<br>");
                         out.println(" Haga click en la imagen para
                          regresar a la p&aacute;gina anterior.");
                          out.println("</div>");
                          out.println("</form>");
                          out.println("</body></html>");
                }
    }
        } catch(NullPointerException npe)
        {
           /*este error se genera la primera vez que se invoca al jsp
           debido al resultado de los metodos request.getParameter
           */
           out.println("JSP:login. Ocurrio el error"+npe.getMessage());
%>
           <%@ include  file="../html/login.html" %>
<%      }
                                                                                    %>
```

APÉNDICE O

Enterprise JavaBeans

Características de los componentes

Las principales características de un componente empresarial son:

- Un componente empresarial típicamente contiene lógica del negocio que opera los datos de la empresa.
- Las instancias de un componente empresarial son creadas y manejadas en tiempo de ejecución por un contenedor (*EJB Container*).
- Varios servicios de información, tales como los atributos de transacción y de seguridad, son separados de las clases de los componentes empresariales. Esto permite a los servicios de información ser manejados por herramientas durante la integración de la aplicación y su publicación.
- El acceso del cliente es por medio del contenedor (*Container*) en el cual el componente empresarial ha sido publicado.
- Si un componente empresarial utiliza solamente los servicios definidos en la especificación EJB, el componente empresarial puede ser publicado en cualquier contendor de EJB (*EJB Container*) disponible. Los contenedores especializados pueden proporcionar servicios adicionales más alla de aquellos definidos en la especificación de EJB. Un componente empresarial que dependa de algún servicio adicional puede ser publicado solamente en un contenedor que soporte ese servicio.
- Un componente empresarial puede ser incluido en la integración de una aplicación sin requerir el cambio o recompilación en su código fuente.

Arreglos de presentación al cliente para componentes de sesión y de entidad

Los arreglos de presentación al cliente son arreglos entre un cliente y un contenedor. Los arreglos de presentación al cliente proporcionan un modelo de desarrollo uniforme para aplicaciones usando componentes empresariales como componentes. Este modelo uniforme permite el uso de herramientas de desarrollo de alto nivel y permite el reuso de una mayor parte de componentes.

El cliente de un componente empresarial puede ser otro componente empresarial publicado en el mismo o diferente contenedor. Puede ser también un programa Java cualquiera, tal como una *aplicación*, *applet*, o *servlet*.

El cliente de un componente de entidad o de sesión puede ser un cliente remoto o puede ser un cliente local.

El arreglo de presentación de un componente empresarial remoto es remoto, ambos programas remotos y locales pueden accesar un componente empresarial utilizando las misma vista remota del componente empresarial. Los objetos que utilizan las interfaces para la presentación del cliente remoto son objetos remotos Java y son accesibles a un cliente a través las API's estándar de Java para invocación de objetos remotos.

El arreglo de presentación de un componente empresarial local no es remoto, solamente los componentes locales pueden accesar al componente empresarial a través de la vista local del cliente. Los componentes locales se ejecutan en la misma máquina virtual Java (JVM) para accesar localmente al componente empresarial [SUNEJB].

Existen dos tipos de Enterprise Java Bean (EJB), *entity beans* (componentes de entidad) y los *session beans* (componentes de sesión).

Componentes de entidad (Entity beans)

Un componente de entidad es un objeto con las siguientes propiedades:

- **Permanente**

  Los objetos Java estándar existen cuando son creados en un programa. Cuando el programa termina, el objeto se pierde. Pero un componente de la entidad permanece alrededor hasta que es borrado. Un programa puede crear un componente de entidad, luego el programa puede ser detenido o reestablecido. El componente de entidad continuará existiendo. Después de ser reestablecido, el programa puede de nuevo encontrar el componente de entidad con el que estaba trabajando y continuar utilizando el mismo componente de entidad.

- **Basado en red**

  Los objetos Java estándar son utilizados solamente por un programa. Pero un componente de entidad puede ser utilizado por cualquier programa en la red. El programa solamente necesita poder encontrar el componente de entidad para utilizarlo.

- **Se ejecuta remotamente**

  Los métodos de un componente de entidad se ejecutan en el servidor. Cuando se invoca un método de un componente de entidad, los hilos del programa detendrán su ejecución y el control pasará del lado del servidor. Cuando el método regresa del servidor, el hilo local comienza de nuevo a ejecutase.

- **Indentificado por una llave primaria**

  Los componentes de entidad deben de tener una llave primaria. La llave primaria es única. Solamente se puede utilizar un componente de entidad cuando sus objetos tienen un campo identificador único, o cuando pueda agregar tal campo.

Componentes de sesión (Session beans)

Los componentes de sesión son diferentes que los componentes de entidad y es que ellos no son objetos permanentes. Tampoco son compartidos en general, aunque es posible compartilos manipulándolos.

Los componentes de sesión pueden ser utilizados para distribuir o aislar tareas de procesos, algo análogos en la manera en que las clases de Java pueden ser utilizadas para encapsular un tipo de proceso relacionado. Cada componente de sesión puede ser utilizado para desarrollar una tarea especifica a nombre de su cliente. La tareas pueden ser distribuidas en diferentes máquinas.

Otras forma de componentes de sesión pueden estar pensadas, tal y como los navegadores y *web servers* trabajan. Un web server esta ubicado en un lugar particular, pero multiples browsers pueden conectase y obtener la ejecución de servicios (tal como la entrega de páginas HTML) para su beneficio.

Ejemplo de código fuente de Enterprise JavaBean

En el desarrollo de este sistema se hizo uso de la arquitectura EJB para hacer más robustos y funcionales los procesos más críticos e importantes del sistema; por ejemplo, el cobro de la orden de contratación. A continuación presentó un fragmento del código de un EJB desarrollado en el sistema:

Nombre del archivo: CargosRechazados
Propósito: Identifica el rechazo de la petición al servidor de banco para realizar el cargo de una tarjeta de crédito.

```
<%@ page import="java.util.*" %>
<%@ page import="java.text.*" %>
<%@ page import="java.sql.*" %>
<%@ page import="javax.naming.*" %>
<%@ page import="java.net.*" %>
<%@ page import="javax.rmi.*" %>
<%@ page import="java.rmi.RemoteException" %>
<%@ page import="javax.ejb.*" %>
<%@ page import="RegistroCobrosBeans.*" %>
<%@ page import="classes.Pago" %>
<%@ page import="classes.*" %>
<%@ page import="sockets.*" %>
<%@ page isThreadSafe="false" %>
<%!
   private String  vcompania  = new String();
   private String  vzona      = new String();
   private String  vplaza     = new String();
   private String  vretail    = new String();
   private String  vuser      = new String();
   private String  vextra     = new String();
   private String  vusuapli   = new String();
   private String  vqueue     = new String();

   private String  dbName     = new String("jdbc/Cobranzas");
   private String  vmoneda    = new String(" ");
   private String  vfpa       = new String(" ");
   private String  vespecie   = new String(" ");
   private float   vimporte     = 0;
   private String  vticket    = new String(" ");
   private String  vbanco     = new String(" ");
   private String  vcuenta    = new String(" ");
   private int     vmes         = 0;
   private int     vanio        = 0;

   private int     vconsecutivo = -1;
   private int     vlength      = 0;
   private int     vsize        = 0;
   private int     vborrar      = 0;
   private int     vresult      = 0;

   private String  vforMoneda        = new String();
   private float   vtotalFP     = 0;
   private float   vtotalEfe    = 0;
   private float   vtotalMn     = 0;
   private float   vtotalUs   = 0;

   private Vector  vmonedas     = new Vector();
   private Vector  vformasPago  = new Vector();
   private Vector  vpagoEspecie = new Vector();
```

```
   private Vector  vbancos      = new Vector();
   private Vector  vpagos       = new Vector();
   private Vector  vMonTarjeta;
   private Vector  vEspTarjeta;
   private Vector  vMonEfe;
   private Vector  vTicket;

   private Pago    vpago;

   private String  vnumAut     = new String();
   private String  vnomAut     = new String();
   private String  vpathEx     = new String();

   private String  verrorMess  = new String();
   private int           verror = 0;

   private String  vcadena     = new String();
   private String  vtipoFP     = new String();
   private String  vtipoFPEsp  = new String();

   private int     posicionini = 0;
   private int     posicionfin = 0;

   private String  vmensaje    = new String();

   private RegistroCobros regCobros;

   private String  vErrorDesc = new String();
   private String  vErrorSecc = new String();
   private String  vModulo    = new String("Cobranzas");
   private String  vPrograma  = new String("FrameDetalle.jsp");

public synchronized void jspInicio(HttpServletRequest request, HttpSession session) {

    /* Obtenemos el Contexto del Servlet para obtener el EJB (en caso de existir) */
    //regCobros = (RegistroCobros)application.getAttribute("simplesession.regCobros");
    if (regCobros == null) {
       System.out.println("NO EXISTE CONEXION EN EL DETALLE");}
    else{
       System.out.println("RECUPERANDO BEAN EN DETALLE");}
}

public void jspDestroy(){
       regCobros = null;
}

public synchronized String eliminaComas(String vstring){

    while (vstring.indexOf(",") > 0){
       vstring = vstring.substring(0,vstring.indexOf(",")) +
vstring.substring(vstring.indexOf(",") + 1,vstring.length());}
    System.out.println("CADENA SIN COMAS: " + vstring);
    return vstring;
}

public synchronized String eliminaDecimal(String vstring){
   int    numDec = 0;
   String vdec = null;
   String vstrTmp = null;

   // Validamos k se tengan 2 decimales
   vstring = vstring.trim();
   vdec = vstring.substring(vstring.indexOf(".")+1,vstring.length());
   vdec = vdec.trim();
   numDec = vdec.length();
   System.out.println("Numero de decimales: " + numDec);

   if (numDec < 2 || numDec > 2){
       if (numDec == 0)
           vstring = vstring + "00";
       else if (numDec == 1)
```

```
           vstring = vstring + "0";
        else if (numDec > 2)
           vstring = vstring.substring(0,vstring.indexOf(".")+3);}

        //quitamos punto decimal
        vstring = vstring.substring(0,vstring.indexOf(".")) +
        vstring.substring(vstring.indexOf(".") + 1,vstring.length());
        System.out.println("CADENA SIN PUNTO DECIMAL: " + vstring);
        return vstring;
}

public synchronized String getParam(String vparam, String vcom, String vuser, String vpass){

    String vresult = new String("0");
    String vstring = new String("");

    try{
        System.out.println("Parametro: " + vparam);
        System.out.println("Compania : " + vcom);
        System.out.println("Usuario  : " + vuser);
        System.out.println("Password : " + vpass);

        System.out.println("FrameDetalle.jsp.getParam()");

        vstring = "PRC_OBTEN_PARAMETROS";

        /* Procedimiento a llamar */
        String vquery  =      "{call " + vstring + "( ?, ?, ? )}";

        System.out.println("Llamada al procedimiento: " + vquery);

        CDBManager dbm = null;
        Cifrar llave   = new Cifrar();

        try{
            dbm    = new CDBManager(dbName);

            if (!dbm.createConnection(vuser,llave.decodificar(vpass))){
                System.out.println("CONEXION FALLIDA CON BD !!!");
                //throw new Exception(dbm.getError());
                return null;
            }

        Statement stmt          = dbm.getStatement();
        java.sql.Connection cnn = dbm.getConnection();

        CallableStatement cstmt;

        cstmt   =      cnn.prepareCall(vquery);

        cstmt.setString( 2, vcom ); // compania
        cstmt.setString( 3, vparam ); // parametro

        System.out.println("Compania:    " + vcom);
        System.out.println("Param   :    " + vparam);

        // se registran parametros de salida OUT
        cstmt.registerOutParameter( 1, Types.VARCHAR );

        //System.out.println("Se establece parametro de salida.");

        //se ejecuta el procedimiento
        cstmt.execute();

        // obtener los valores de salida
        vresult = cstmt.getString( 1 );
        System.out.println("Salida  : " + vresult);

        }catch(SQLException sqle){
               /* Capturar Error */
               vErrorDesc = sqle.getMessage();
```

```
                vErrorSecc = "10810";
                System.out.println("Error en: Seccion[" + vErrorSecc + "] / " + vErrorDesc +
                " / " + vModulo + " / " + vPrograma );

                return null;
        }catch(Exception e){
                /* Capturar Error */
                vErrorDesc = e.getMessage();
                vErrorSecc = "10820";
                System.out.println("Error en: Seccion[" + vErrorSecc + "] / " + vErrorDesc +
                " / " + vModulo + " / " + vPrograma );

                return null;
        }finally{
                dbm.closeConnection();
                //closeConnection();
        }


        }catch(Exception e){
                /* Capturar Error */
                vErrorDesc = e.getMessage();
                vErrorSecc = "10830";
                System.out.println("Error en: Seccion[" + vErrorSecc + "] / " + vErrorDesc +
                " / " + vModulo + " / " + vPrograma );
                return null;
        }
        return vresult;
}

public synchronized void InsertFailedCC ( String retail,
                                          String tipoDoc,
                                          String numDoc,
                                          String tarjeta,
                                          String fecha,
                                          float  importe,
                                          float  tipoCambio,
                                          int    codMens,
                                          String mens,
                                          String vuser,
                                          String vpass){

        int    vresultCod = 0;
        String  vresultMsg = new String("0");
        String  vstring    = new String("");

        System.out.println("Retail:      " + retail);
        System.out.println("Tipo Doc:    " + tipoDoc);
        System.out.println("No. Doc:     " + numDoc);
        System.out.println("Tarjeta:     " + tarjeta);
        System.out.println("Fecha:       " + fecha);
        System.out.println("Importe:     " + importe);
        System.out.println("Tipo Cambio: " + tipoCambio);
        System.out.println("Codigo Resp: " + codMens);
        System.out.println("Mensaje:     " + mens);
        System.out.println("Usuario:     " + vuser);
        System.out.println("Password:    " + vpass);

        try{

            System.out.println("FrameDetalle.jsp.InsertFailedCC ()");

            vstring = "PRC_CREDIT_TRANSACTION_REFUSED";

            /* Procedimiento a llamar */
            String vquery ="{call " + vstring + "( ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?  )}";

            System.out.println("Llamada al procedimiento: " + vquery);

            CDBManager dbm = null;
            Cifrar llave = new Cifrar();
```

```
        try{

            dbm    = new CDBManager(dbName);

            if (!dbm.createConnection(vuser,llave.decodificar(vpass))){
                System.out.println("CONEXION FALLIDA CON BD !!!");
                //throw new Exception(dbm.getError());
                 return;}

            Statement stmt = dbm.getStatement();
            java.sql.Connection cnn      = dbm.getConnection();

            CallableStatement cstmt;

            cstmt = cnn.prepareCall(vquery);

            cstmt.setString( 1, retail );     // retail
            cstmt.setString( 2, tipoDoc );    // tipo documento
            cstmt.setString( 3, numDoc );     // numero documento
            cstmt.setString( 4, tarjeta );    // tarjeta
            cstmt.setString( 5, fecha );      // fecha
            cstmt.setFloat( 6, importe );     // importe
            cstmt.setFloat( 7, tipoCambio );  // tipoCambio
            cstmt.setInt( 8, codMens );       // codigo Mensaje
            cstmt.setString( 9, mens );       // mensaje
            cstmt.setString(10, vuser );      // usuario

            // se registran parametros de salida OUT
            cstmt.registerOutParameter( 11, Types.INTEGER );

            //se ejecuta el procedimiento
            cstmt.execute();

            // obtener los valores de salida
            vresultCod = cstmt.getInt( 11 );
            System.out.println("Salida  : " + vresultCod );

            }catch(SQLException sqle){
                   /* Capturar Error */
                   vErrorDesc = sqle.getMessage();
                   vErrorSecc = "10840";
                   System.out.println("Error en: Seccion[" + vErrorSecc + "] / " +
                   vErrorDesc + " / " + vModulo + " / " + vPrograma );

                   //vresult = null;
            }catch(Exception e){
                   /* Capturar Error */
                   vErrorDesc = e.getMessage();
                   vErrorSecc = "10850";
                   System.out.println("Error en: Seccion[" + vErrorSecc + "] / " +
                   vErrorDesc + " / " + vModulo + " / " + vPrograma );

                   //vresult = null;
            }finally{
                   dbm.closeConnection();
                   //closeConnection();
            }

       }catch(Exception e){
              /* Capturar Error */
              vErrorDesc = e.getMessage();
              vErrorSecc = "10860";
              System.out.println("Error en: Seccion[" + vErrorSecc + "] / " + vErrorDesc +
              " / " + vModulo + " / " + vPrograma );
              //vresult = null;
       }

}
%>
```

APÉNDICE Q

Java Message Service

Java Message Service (*JMS*) proporciona una manera común en programas Java para crear, enviar, recibir y leer un mensaje empresarial en un sistema de mensajería.

Los productos de mensajería empresarial han llegado a ser un componente esencial para la integración de operaciones intra-compañías. Ellos permiten separar componentes de negocios para ser combinados en un confiable y flexible sistema.

El cliente del lenguaje Java y el lenguaje Java en su servicio de capa media deben ser capaces de utilizar esos sistemas de mensajería. *JMS* proporciona una manera común en programas Java para accesar a esos sistemas.

Java Message Service es un conjunto de interfaces y semánticas asociadas que define cómo un cliente JMS aprovecha las facilidades de un producto de mensajería empresarial.

Objetivos de JMS

Java Message Service define un conjunto de facilidades y conceptos de mensajería empresarial comunes. Intenta minimizar el conjunto de conceptos que un programador de lenguaje Java debe aprender para utilizar productos de mensajería empresarial. Se esfuerza por maximizar la portabilidad de aplicaciones de mensajería.

¿ Qué es una aplicación JMS ?

Una aplicación JMS esta compuesta de las siguientes partes:

- Clientes JMS — Son los programas en lenguaje Java que envían y reciben mensajes.
- Mensajes — Cada aplicación define un conjunto de mensajes que son utilizados para comunicar información entres sus clientes.
- Proveedor JMS — La mayoría de los servidores de aplicaciones cuentan con contenedores que proporcionan el servicio de mensajería.
- Objetos administrados — Son objetos JMS preconfigurados creados por un administrador para el uso de los clientes.

Administración

Es evidente que los proveedores de JMS diferirán significativamente en su entendimiento de la tecnología de mensajería. Es también evidente que existirán mayores diferencias de cómo el sistema del proveedor es instalado y administrado.

Si los clientes JMS son portables, ellos pueden ser apartados de esos aspectos de propiedad del proveedor. Esto es hecho definiendo objetos administrados JMS que son creados y modificados por un administrador del proveedor y más tarde utilizados por los clientes. Los clientes los utilizan a través de interfaces JMS que son portables. El administrador los crea utilizando las facilidades especificas del proveedor.

Existen dos tipos de objetos administrados:

- ConnectionFactory — Es el objeto que el cliente utiliza para crear una conexión con un proveedor.
- Destination — Este es un objeto que el cliente utiliza para especificar el destino de un mensaje que se está enviando y la fuente del mensaje recibida.

Los objetos administrados son ubicados en un JNDI por un administrador. Un cliente JMS comúnmente nota en su documentación el objeto JMS administrado que requiere y cómo el nombre del JNDI de esos objetos sería proporcionado.

Interfaces JMS

Java Message Service esta basado en un conjunto de conceptos de mensajería en común. Cada dominio de mensaje JMS Point-to-Point y Publish / Subscriber también definen un conjunto de interfaces a la medida para esos conceptos.

Interface Común JMS	Interface Point-to-Point	Interface Publish / Subscriber
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Destination	QueueDestination	TopicDestination
Session	QueueSession	TopicSession
MessageProducer	QueueMessageProducer	TopicMessageProducer
MessageConsumer	QueueMessageConsumer	TopicMessageConsumer

Relación de interfaces Point-to-Point y Publish / Subscriber

Las interfaces comunes de JMS proporcionan un dominio independiente de los dominios de los mensajes Point-to-Point y Publish / Subscriber.

Interface Común JMS	Descripción
ConnectionFactory	Objeto administrado por un cliente para crear una conexión.
Connection	Conexión activa a un proveedor JMS.
Destination	Objeto administrado que encapsula la identidad del destino del mensaje.
Session	Contexto de un solo hilo para enviar y recibir mensajes.
MessageProducer	Objeto creado por una sesión que es utilizada para enviar mensajes a un destino.
MessageConsumer	Objeto creado por una sesión que es utilizada para recibir mensajes a un destino.

Relación de objetos de JMS

Relación de JMS con otras API's de Java

- **Java DataBase Connectivity (JDBC)**

  Los clientes JMS también pueden utilizar el API JDBC. Pueden desear incluir el uso de ambas API's JDBC y JMS en la misma transacción. En la mayoría de los casos, esto será logrado automáticamente por la implementación de esos clientes como componentes Enterprise JavaBeans (EJB).

Modelo de Componente Enterprise JavaBeans

El API de JMS es un recurso importante disponible para desarrolladores de componentes Enterprise JavaBeans. Puede ser utilizado en conjunción con otros recursos como JDBC para implementar servicios empresariales.

La especificación de EJB 2.0 define componentes que son invocados sincronizadamente vía métodos llamados desde clientes EJB. También define una forma de componente asíncrono que es invocado cuando un cliente JMS envía un mensaje, llamada componente de mensaje-conducido (message-driven). La especificación de EJB 2.0 soporta el manejo de ambos mensajes síncronos y asíncronos.

Java Naming and Directory Interface (JNDI)

Los clientes JMS buscan objetos JMS configurados utilizado el API JNDI. Los administradores de JMS utilizan las facilidades de un proveedor específico para crear y configurar esos objetos.

Esta división de trabajo maximiza la portabilidad de clientes delegando trabajo al administrador de un proveedor específico. Esto también conduce a aplicaciones más administrables por que los clientes no necesitan insertar valores administrativos en su código.

Java 2 Enterprise Edition (J2EE)

La especificación de la plataforma J2EE requiere soporte para el API de JMS como parte de la plataforma J2EE. La especificación de la plataforma J2EE ubica ciertos requerimientos adicionales sobre la implementación de JMS mas allá de los descritos en la especificación de JMS, incluyendo el soporte para los dominios point-to-point y publish/subscriber.

Integración de JMS con los componentes EJB

La plataforma J2EE y la especificación EJB describen requerimientos adicionales para un proveedor JMS que es integrado dentro de una plataforma J2EE. Una de las claves en el conjunto de requerimientos es cómo la producción de un mensaje JMS y el consumo de un mensaje JMS interactuan con los requerimientos transaccionales del administrador del contenedor en un componente empresarial.

Esta especificación del API de JMS no hace referencia a un modelo para implementar esos modelos para la integración. Por lo tanto, las diferentes implementaciones de proveedores JMS pueden implementar la integración con la plataforma J2EE y soportar los requerimientos de los EJB en diferentes maneras.

Modelo de Mensajes JMS

Los productos de mensajería empresarial tratan los mensajes como entidades ligeras que consisten de un encabezado (header) y un cuerpo (body). El encabezado contiene campos utilizados para el enrutamiento e identificación del mensaje; el cuerpo contiene la información que esta siendo enviada.

Sin esta forma general, la definición de un mensaje variaría significativamente entre los productos. Existen mayores diferencias en la semántica y contenido de los encabezados.

Mensajes JMS

Los mensajes JMS están compuestos de las siguientes partes:

- Encabezado (header)
  Todos los mensajes soportan el mismo conjunto de campos del encabezado. Los campos del encabezado contienen valores utilizados por ambos clientes y proveedores para identificar y direccionar los mensajes.

- Propiedades (properties)
  En referencia al estándar de los campos del encabezado, los mensajes proporcionan la facilidad de incluir campos de encabezado opcionales a un mensaje.
  - Application specific properites. Esto proporciona un mecanismo para agregar campos de encabezado específicos de la aplicación a un mensaje.
  - Standard properties. JMS define algunas propiedades estándar que son, en efecto, campos opcionales del encabezado.
  - Provider specific properties. La integración de un cliente JMS con un proveedor nativo de JMS puede requerir el uso de propiedades específicas del proveedor. JMS designa una convención para eso.

- Cuerpo (body). JMS define algunos tipos de cuerpo de mensaje los cuales cubren la mayoría de los estilos de mensajes actualmente utilizados.

Campos del encabezado del mensaje

La siguiente tabla describe cada campo del encabezado del mensaje de JMS. El encabezado completo de un mensaje es transmitido a todos los clientes JMS que reciben el mensaje.

- **JMSDestination**

  Contiene el destino al cual el mensaje esta siendo enviado. Cuando un mensaje es enviado, este campo es ignorado. Después terminado el envio, el objeto especificado de destino es llenado por el método de envio. Cuando un mensaje es recibido, su valor de destino debe ser equivalente al valor asignado cuando fue enviado.

- **JMSDeliveryMode**

  Contiene el modo especificado de entrega cuando el mensaje fue enviado. Cuando un mensaje es enviado, este campo es ignorado. Después terminado el envio, el objeto especificado de destino es llenado por el método de envio.

- **JMSMessageId**

  Contiene el valor que identifica como único cada mensaje enviado por un proveedor. Cuando un mensaje es enviado, este campo es ignorado. Cuando el método de envio regresa, el campo contiene un valor asignado por el proveedor. Un JMSMessageId es un valor String el cual funcionará como una llave única para identificar los mensajes en un repositorio de mensajes almacenados.

- **JMSTimestamp**

  Contiene el tiempo que un mensaje fue manejado por un proveedor para ser enviado. Este no es el tiempo en que el mensaje fue transmitido porque el envio actual puede ocurrir después debido a transacciones u otros mensajes encolados del lado del cliente. Cuando un mensaje es enviado, el JMSTimestamp es ignorado. Cuando el método regresa, el campo contiene un valor en algún lugar del intervalo entre la llamada y el regreso.

- **JMSCorrelationId**

  Un cliente puede utilizar el campo de encabezado JMSCorrelationId para ligar un mensaje con otro. Un típico uso es ligar una respuesta con su petición.

- **JMSReplayTo**

  Contiene el destino proporcionado por un cliente cuando un mensaje es enviado.

- **JMSDelivered**

  Si un cliente recibe un mensaje con el indicador JMSDelivered, es probable, pero no seguro, que ese mensaje fue entregado pero no aceptado anteriormente.

- **JMSType**

  Contiene un tipo de identificador del mensaje proporcionado por un cliente cuando es enviado. Algunos proveedores de JMS usan un repositorio de mensajes que contienen las definiciones de los mensajes enviados por aplicaciones. El campo identificador *type* puede referenciar la definición de esos mensajes en el repositorio del proveedor.

- **JMSExpiration**

  Cuando un mensaje es enviado, su tiempo de expiración es calculado como la suma de los tiempos de vida (*time-to-live*) especificados en el método de envio. En el regreso del método de envio, el campo de encabezado JMSExpiration del mensaje contiene este valor. Cuando un mensaje es recibido su campo de encabezado JMSExpiration contiene este mismo valor. Si el tiempo de vida es especificado como cero, la expiración es puesta a cero para indicar que el mensaje no expira.

- **JMSPriority**

  Contiene la prioridad del mensaje. Cuando un mensaje es enviado, este campo es ignorado. Después de completado el mensaje, contiene el valor especificado por el método de enviar mensajes. JMS define diez niveles como valores de prioridad, con 0 como el valor más bajo y 9 como el valor más alto. Además, los clientes deberán considerar las prioridades 0-4 como graduaciones normales y las prioridades 5-9 como graduaciones de prioridad alta.

Como los valores de encabezado de mensajes son puestos

Campo de Encabezado de Mensaje	Valor asignado por
JMSDestination	Método de enviar
JMSDeliveryMode	Método de enviar
JMSMessageId	Método de enviar
JMSTimestamp	Método de enviar
JMSCorrelationId	Método de enviar
JMSReplayTo	Método de enviar
JMSDelivered	Cliente
JMSType	Cliente
JMSExpiration	Cliente
JMSPriority	Proveedor

Cuerpo del Mensaje

Java Message Service proporciona cinco forma de cuerpo de mensaje. Cada forma es definida por una interfaz:

- StreamMessage — Un mensaje cuyo cuerpo contiene un valor primitivo de Java de tipo *stream*. Este es llenado y leído secuencialmente.
- MapMessage — Un mensaje cuyo cuerpo contiene un conjunto de valores pares nombrados donde los nombres cadenas y valores son tipos primitivos de Java. Las entradas puedes ser ingresadas secuencialmente indice o aleatoriamente por nombre. El orden de las entradas es indefinido.
- TextMessage — Un mensaje cuyo cuerpo contiene un java.lang.String. La incorporación de este tipo de mensaje esta basado en la suposición de que el mensaje String será utilizado extensamente. Una razón para esto es que XML probablemente llegará a ser un mecanismo para representar el contenido de un mensaje JMS.
- ObjectMessage — Un mensaje que contiene un *objeto Java serializable*. Si una colección de objetos java es necesitada, una de las colecciones de clases proporcionadas en el JDK pueden ser utilizadas.
- BytesMessage — Un mensaje que contiene un stream de bytes no interpretados.

Tipos de mensajes

Modelo point-to-point

Los sistemas point-to-point son aquellos que trabajan con colas de mensajes. Ellos son point-to-point porque el cliente envia un mensaje a un cola específica. Es común para un cliente tener todos sus mensajes entregados a una sola cola.

El modelo point-to-point de JMS define cómo un cliente trabaja con sus colas: cómo las encuentra, cómo les envía mensajes y cómo recibe mensajes de ellas.

La siguiente tabla muestra las interfaces específicas para el dominio point-to-point.

Interfaces de Dominio PTP	Descripción
QueueConnectionFactory	Encapsula un conjunto de parámetros de configuración para conexiones que han sido definidos por un administrador. Un cliente lo utiliza para crear un Connection con un proveedor JMS.
QueueConnection	Es la conexión de un cliente activo con su proveedor JMS. Típicamente representa la apertura TCP/IP de un *socket* entre un cliente y el demonio de servicios de un proveedor.
Queue	Encapsula el nombre de una cola específica del proveedor. Es la forma en que un cliente especifica la identidad de la cola a un método JMS.
QueueSession	Es el contexto de un solo hilo para enviar y recibir mensajes.

QueueSender	Un cliente utiliza un QueueSender para enviar mensajes a un destino. Un cliente puede especificar un modo de entrega, prioridad y tiempo de vida por default para los mensajes enviados por un productor. También puede especificar modo de entrega, prioridad y tiempo de vida por default para cada mensaje.
QueueReceiver	Un cliente utiliza un QueueReceiver para recibir mensajes de un destino.

Modelo publish-subscribe

El modelo publish-subscribe (publicar-subscribir) define como clientes de JMS publican mensajes y suscriben mensajes de un bien conocido nodo en un contenido jerárquico. JMS llama a esos nodos temas (*topics*).

Un tema puede ser concebido como un intermediario de mini-mensajes que junta y distribuye mensajes dirigidos a él. Para confiar en el tema como un intermediario, los mensajes publicados son almacenados independientemente de los suscritos y viceversa. Los publicadores y suscriptores están activos cuando los objetos Java que representan existen.

La siguiente tabla muestra las interfaces específicas para el dominio publish / subscriber.

Interfaces de Dominio P / S	Descripción
TopicConnectionFactory	Encapsula un conjunto de parámetros de configuración para conexiones que han sido definidos por un administrador. Un cliente lo utiliza para crear un Connection con un proveedor JMS.
TopicConnection	Es la conexión de un cliente activo con su proveedor JMS. Un cliente utiliza TopicConnection para crear una o más TopicSessions para enviar y recibir mensajes.
Topic	Encapsula el nombre de un tema (*topic)* específico del proveedor. Es la forma en que un cliente especifica la identidad de un *topic* a un método JMS.
TopicSession	Proporciona métodos para crear TopicPublishers y TopicSubscribers. También proporciona métodos no subscritos para borrar la subscripciones durables del cliente. Si existen mensajes que han sido enviados pero no notificados cuando un TopicSession termina, un TopicSubscriber durable debe retenerlos y re-entregarlos. Un TopicSubscriber no durable no necesita hacer eso.
TopicPublisher	Un cliente utiliza TopicPublisher para publicar mensajes sobre un *topic*.
TopicSubscriber	Un cliente utiliza un TopicSubscriber para recibir mensajes que han sido publicados en un *topic*. Normalmente un TopicSubscriber no es durable. Solamente reciben mensajes que son publicados mientras están activos.

Ejemplo de código fuente de Java Message Service

A continuación muestro el código fuente de uno de los Jave Service Message que se realizaron en el Sistema de Comercialización en Línea:

Nombre del archivo: EnviaCredito
Propósito: Realiza el envio del mensaje (Orden de Contratación) a la cola de mensajes del sistema.

```
package jms;


import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.naming.*;
import javax.jms.*;
import java.util.*;

import classes.*;

public class EnviaCredito extends HttpServlet {

    public void init() throws ServletException {

   }

   public void doGet (HttpServletRequest req, HttpServletResponse res)
                     throws ServletException, IOException{
       performTask (req,res);
   }

   public void doPost (HttpServletRequest req, HttpServletResponse res)
                                  throws ServletException, IOException{
       performTask (req,res);
   }

   public void performTask (HttpServletRequest request, HttpServletResponse response) throws
                                          ServletException, IOException{
       response.setContentType("text/html");
       PrintWriter out = response.getWriter();

       try{
           /* Obtener session */
           HttpSession session= request.getSession ();
           Domicilio vdomTemp = new Domicilio();
           out.println("SE ENVIAN PARAMETRO POR JMS");
           vdomTemp.setCalle("Ing. Marroquin y Rivera #29");
           vdomTemp.setColonia("Guadalupe Insurgentes");
           vdomTemp.setEstado("Mexico D.F.");
           /*if (request.getParameter("venvia")) != null{
           }*/
           out.println("Calle  : " + vdomTemp.getCalle());
           out.println("Colonia: " + vdomTemp.getColonia());
           out.println("Estado : " + vdomTemp.getEstado());

           /*Hashtable hSession= new Hashtable ();
           hSession.put ("User", (String) session.getAttribute ("User"));
           hSession.put ("Pass", (String) session.getAttribute ("Pass"));

           hSession.put ("usu_ret_pla_zon_com_clave", (String) session.getAttribute
```

```
            ("usu_ret_pla_zon_com_clave"));
           hSession.put ("usu_ret_pla_zon_clave", (String) session.getAttribute
           ("usu_ret_pla_zon_clave"));
           hSession.put ("usu_ret_pla_clave", (String) session.getAttribute
           ("usu_ret_pla_clave"));
           hSession.put ("usu_ret_clave", (String) session.getAttribute ("usu_ret_clave"));
           hSession.put ("usu_rol_clave", (String) session.getAttribute ("usu_rol_clave"));
           hSession.put ("usu_aplicacion", (String) session.getAttribute
           ("usu_aplicacion"));


           Hashtable hDocument= new Hashtable ();
           hDocument = (Hashtable) session.getAttribute ("HDOCUMENT");
           request.setAttribute ("HDOCUMENT",hDocument);
           Hashtable hData=new Hashtable ();
           hData.put ("HSESSION",hSession);
           hData.put ("HDOCUMENT",hDocument);*/
           Context jndiContext = new InitialContext();
           /* Se localiza el CONECTION FACTORY y la QUEUE */
           QueueConnectionFactory  queueConnectionFactory = (QueueConnectionFactory)
           jndiContext.lookup("java:comp/env/jms/QueueCredit");
           Queue queue = (Queue) jndiContext.lookup("java:comp/env/jms/EnvioSolicitud");
           /* Se crea la QUEUE de Coneccion, SESSION y el SENDER */
           QueueConnection queueConnection = queueConnectionFactory.createQueueConnection();
           QueueSession queueSession =
           queueConnection.createQueueSession(false,Session.AUTO_ACKNOWLEDGE);
           QueueSender queueSender = queueSession.createSender(queue);

           /* Se envian mensajes a la QUEUE */
           ObjectMessage   message = queueSession.createObjectMessage(); // tipo de Mensaje
           message.setObject ("mensaje"); // se inserta mensaje
           queueSender.send(message); // se envia el mensaje


           /*String page="/../contratacion/jsp/SavedOrdenEjms.jsp";
           System.out.println (page);
           ServletContext sc = getServletConfig().getServletContext();

           RequestDispatcher rd = sc.getRequestDispatcher(page);
           rd.forward(request,response); */

           /* Se cierran las Conexiones a la Queue */
           queueSession.close ();
           queueConnection.close ();
           queueSender.close ();

           message=null;
           queueSession=null;
           queueConnection=null;
           queueSender=null;
           queueConnectionFactory=null;

           }
           catch (JMSException e) {
              e.printStackTrace ();
              System.out.println(e.getMessage());
           }
           catch (NamingException e) {
              e.printStackTrace ();
              System.out.println(e.getMessage());
           }
out.close();
   }

}
```

APÉNDICE S

Mejoras al sistema

El desarrollo de sistemas hoy por hoy se ve fortalecido por el uso de nuevas tecnologías; por ejemplo, *workflow*. El término *workflow* es usado para describir la tareas, pasos en el procedimiento, organizaciones o personal involucrado, requerido para accesar o hacer salir información y las herramientas necesarias para cada paso en un proceso de negocio. Una aproximación con *workflow* para analizar y administrar un proceso de negocio puede ser combinado con una aproximación de programación orientada a objetos, los cuales tienden a centrarse en documentos y datos. En general, la administración de *workflow* se enfoca en procesos más que en documentos.

Un motor de *workflow* es el componente en un programa de automatización de *workflow* que sabe todos los procedimientos, pasos en el procidimiento y las reglas para cada paso. El motor de *workflow* determina si el proceso esta listo para ir el siguiente paso. A continuación presento un ejemplo con el diagrama de un proceso de negocio utilizando un *workflow*.

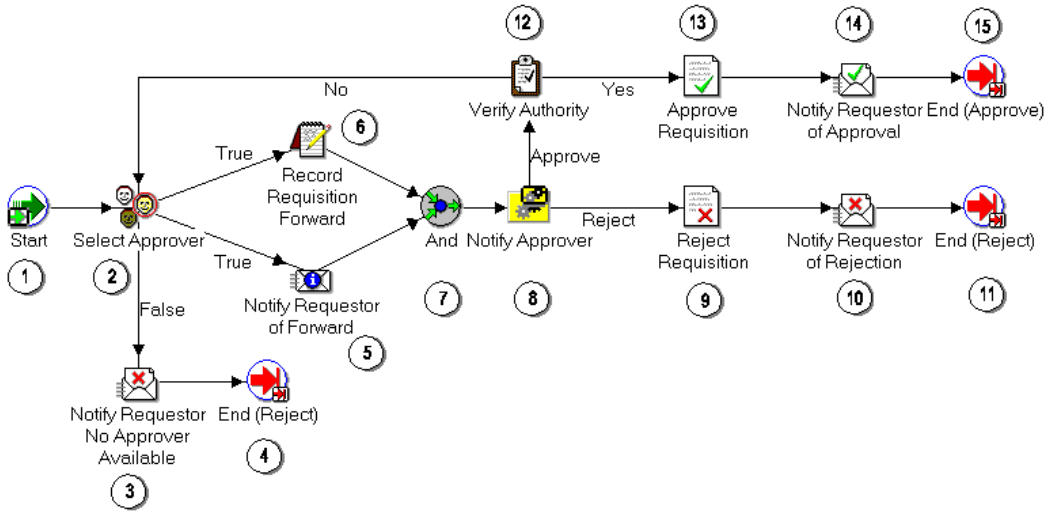


Figura 1. Ejemplo de un diagrama en workflow

Este tipo de tecnología nos puede permitir tener un mayor control en el diseño y desarrollo de un sistema, ya que se tienen perfectamente identificados los pasos en el flujo de un proceso de negocio. Así mismo se puede estandarizar la documentación técnica del proyecto, lo cual es una parte muy importante en el momento que el sistema requiere mantenimiento.

APÉNDICE T

Glosario de términos

API	Un API (Application Program Interface) es el método específico prescrito por un sistema operativo de la computadora o por un programa de uso por el cual un programador que escribe un programa de uso pueda hacer peticiones al sistema operativo u otra aplicación.
Applet	Un applet es un programa que se puede enviar junto con una página del web a un usuario. Los Java applets pueden realizar animaciones interactivas, cálculos inmediatos, u otras tareas simples sin tener que enviar una petición de usuario de nuevo al servidor.
artefacto	En lo referente a la metodología UML es una información que es utilizada o producida mediante un proceso de desarrollo de software.
bits	Un bit es la más pequeña unidad de datos en una computadora y tiene un solo valor binario un 1 ó 0.
browsers o navegadores	Un navegadores un programa de uso que proporciona una manera de mirar y de obrar recíprocamente con toda la información sobre el web mundial.
buró de crédito	Es una Sociedad de Información Crediticia, su operación y la de los Otorgantes de Crédito que tienen contratado su servicio está regida por la Ley para Regular a las Sociedades de Información Crediticia y por las Reglas Generales que dicta Banco de México.
bytecode	Bytecode es el código de objeto de la computadora que es procesado por un programa, designado generalmente a una máquina virtual, más bien que por la máquina "verdadera" de la computadora, el procesador del hardware. La máquina virtual convierte cada instrucción de máquina generalizada en una instrucción o las instrucciones específica de máquina que el procesador de esta computadora entienda.
caso de uso	Documento narrativo de la metodología UML que describe la secuencia de los eventos de un actor que utiliza un sistema para completar un proceso.
crackers	Un cracker es una persona que logra romper el esquema de seguridad de un sistema informático, a menudo en una red, contraseñas o licencias en programas de computadora
datagramas	El término datagrama ha sido substituido generalmente por el término de paquete. Los datagramas o los paquetes son las unidades del mensaje que el Protocolo de Internet ocupa para transmitir información en Internet.
diagrama de casos de uso	Documento de la metodología UML que explica gráficamente un conjunto de casos de uso de un sistema, los actores y la relación de éstos y los casos de uso.

diagrama de clases	Documento de la metodología UML que muestra gráficamente las especificaciones de las clases de software y la interfaz de la aplicación.
diagrama de secuencia	Documento de la metodología UML que muestra gráficamente los eventos que fluyen de los actores al sistema.
dominio	En lo referente a base de datos, es un conjunto de valores permitidos y reglas de formato de un atributo.
encapsulación	En la programación orientada a objetos, la encapsulación es la inclusión de toda la necesidad de los recursos del objeto a funcionar (básicamente, de los métodos y de los datos) dentro de un objeto del programa.
enterprise beans	Existen tres tipos de *enterprise beans*: *session beans*, *entity beans*, y *message-driven beans*.
entity bean	Un entity beans representa un dato persistente almacenado en un registro de una tabla de base de datos. Si el cliente termina o si el servidor es dado de baja, el servicio asegura que el dato del entity bean sea grabado
evento de un sistema	El evento de un sistema es un hecho externo de entrada que un actor produce en un sistema. El evento da origen a una operación de respuesta.
gateway	Es un hecho externo de entrada que un actor produce en un sistema.
herencia	En la programación orientada a objetos, la herencia es la facilidad de heredar las propiedades de un objeto padre a sus objetos hijos.
host	En la especificación del Protocolo de Internet, el término host significa cualquier computadora que tenga completo acceso de dos vias a otras computadoras en el Internet.
J2EE	Es una plataforma Java diseñada para ser escalada en servidores de grandes empresas.
Java Message Service	JMS es un Application Program Interface (API) de Sun Microsystems que apoya la comunicación formal conocida como mensajería entre las computadoras en una red.
JavaServer Page	JSP es una tecnología de la plataforma Java 2 Enterprise Edition (J2EE), para construir aplicaciones para conternido web dinámicamente generado, tal como HTML, DHTML, XHTML y XML.
JDBC	Java Database Connectivity es una especificación del Application Program Interface (API) para los programas escritos en Java que se conectan en bases de datos populares.
JMS Point-to-Point	Los sistemas point-to-point son aquellos que trabajan con colas de mensajes. Ellos son point-to-point porque el cliente envia un mensaje a un cola específica.
JMS Publish / Subscriber	El modelo publish-subscribe (publicar-subscribir) define como clientes de JMS publican mensajes y suscriben mensajes de un bien conocido nodo en un contenido jerárquico.

JNDI	Java Naming and Directory Interface permite a aplicaciones basadas en Java accesar a múltiples nombramientos y directorios de servicios.
kernel	Es el centro escencial de un sistema operativo de computo, el corazón que proporciona servicios básicos para todas las demás partes del sistema operativo.
message-driven beans	*message-driven bean* combina características de un *session bean* y un escuchador de mensajes Java Message Service (JMS), permitiendo a un componente de negocio recibir mensajes JMS asíncronamente.
metamodelo	Un metamodelo es un modelo que define el lenguaje para expresar otros modelos.
modelo conceptual	Un modelo conceptual es la representación de objetos en un dominio del problema. En UML, se ilustra con un conjunto de diagramas de estructura estática donde no se define ninguna operación.
nivel de reporteo	Especifica el nivel de consulta asociado a cada usuario, éste puede ser a nivel de Compañía, Zona, Plaza o Punto de Venta.
normalización	La normalización es el proceso de organizar los datos en una base de datos.
operación de un sistema	Es una acción que el sistema ejecuta en respuesta a un evento del sistema.
orden de contratación	Es el conjunto de información requerida para la realización de una venta y esta relacionada con información personal del cliente, referencias, detalle de equipos de comunicación a adquirir; tal como modelo, número de serie, tipo de contratación, precio, plan tarifario.
paquetes de equipos	Es la definición de un conjunto de equipos de comunicación con determinadas características que la empresa elabora para su venta.
plan tarifario	Establece el costo mensual de un conjunto establecido de servicios para los equipos de comunicación
polimorfismo	Concepto según el dos o más tipos de objetos pueden responder a un mismo mensaje en formas diferentes.
protocolo de red	El protocolo de red está diseñado para utilizarse en la interconexión de sistemas paket-switched (paquetes-intercambiados) de redes de comunicaciones de computadoras
router	En Internet, un router es un dispositivo o en algunos casos, un software en una computadora que determina el punto siguiente de la red a el cual un paquete deberá ser enviado en dirección a su destino.
Servlet	Un servlet es una tecnología Java basado en componentes web, administrado por un contenedor, que genera contenido dinámico y que es ejecutado en un servidor.
session bean	Un session bean representa una conversasión pasajera con un cliente. Cuando el cliente finaliza la ejecución, el session bean y sus datos desaparecen.

thread	Un thread es una secuencia de respuestas a una fijación inicial del mensaje.
UML	UML es una notación estándar para modelar los objetos del mundo real como un primer paso en desarrollar una metodología de diseño orientada a objetos.
URL	Un URL (Uniform Resource Locator, previously Universal Resource Locator) es al dirección de un archivo (recurso) que puede ser ingresado desde Internet.

APÉNDICE U

Bibliogafía recomendada

La lectura de la bibliografía que a continuación menciono me ayudó a aprender, comprender y aplicar conocimientos del desarrollo humano que me permitieron interactuar de forma proactiva con mi equipo de trabajo.

- Los 7 hábitos de la gente altamente efectiva
  Stephen R. Covey
  Editorial Paidos Mexicana
  Primera edición

La lectura de la bibliografía que a continuación menciono me ayudó a aprender, comprender y aplicar conocimientos para el análisis de problemas y diseño de soluciones con el lenguaje de UML (Unified Model language)

- UML y Patrones
  Introducción al análisis y diseño orientado a objetos
  Graig Larman
  Editorial Prentice Hall

La lectura de la bibliografía que a continuación menciono me ayudó a aprender, comprender y aplicar conocimientos técnicos que me permitieron utilizar de manera más eficaz la tecnología utilizada en el desarrollo del proyecto.

- Oracle 8i Advanced PL/SQL Programming
  Scott Urman
  Editorial Osborne – Mc Graw Hill

- Oracle 8i Application Programming with Java, PL/SQL and XML
  EJB, JDBC, SQLJ, JSP, XSLT, SOAP, BC4J, intermedia, ASP
  John Carnell, Bjarki Hólm, Ann Horton, Kevin Mukhar, Daniel O'Connor, Mario Zucca, Michael Awai, Matthew Bortniker, Jaeda Goodman, Thomas Kyte
  Editorial Wrox Press Ltd.

- Java 2 Curso de Programación
  Francisco Javier Cevallos
  Editorial Alfaomega – Ra Ma

- Servlets y JavaServer Pages Guia Práctica
  Marty may
  Editorial Prentice may
  Primera edición.

APÉNDICE V

Referencias

CLAVE	DESCRIPCIÓN
LARMAN99	CRAIG LARMAN, UML Y PATRONES, PRENTICE HALL, 1999.
OMG99	OBJECT MANAGEMENT GROUP, INC., www.omg.com, 1995.
UNLPALM	http://gidis.ing.unlpalm.edu.ar/uml/uml.html
RATIONAL	www.rational.com
CRANGEL	www.creangel.com/uml
IEEE	http://ieee.udistrital.edu.co/concurso/programacionII/Programacion2/html/c356.html
HALL01	MARTY HALL, SERVLETS Y JAVA SERVER PAGE GUÍA PRÁCTICA, PEARSON EDUCATION, 2001
SUNSER	Danny Coward, Java Servlet Specification Versión , Sun Microsystems
SUNJSP	Mark Roth, Eduardo Pelegri- Llopart, JavaServer Pages Specification Versión , Sun Microsystems
SUNEJB	Linda G. DeMichiel, Enterprise JavaBeans Specification Versión , Sun Microsystems
RFC791	www.faqs.org/rfcs/rfc791.html
KEOGH	Jim Keogh, Manual de referencia J2EE, Mc Graw Hill