# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

## FACULTAD DE INGENIERÍA

# "HERRAMIENTA QUE FACILITA EL DESARROLLO DE LA CAPA DE PERSISTENCIA PARA APLICACIONES JAVA"

**T E S I S**

QUE PARA OBTENER EL TÍTULO DE
**INGENIERO EN COMPUTACIÓN**
P R E S E N T A N:

**ALBERTO BALDERAS PÉREZ**
**ISRAEL NORIEGA GUDIÑO**

**DIRECTOR DE TESIS: ING. ALEJANDRO MANCILLA ROSALES**

**CD. UNIVERSITARIA MEXICO, D.F.** **AGOSTO 2006**

## *AGRADECIMIENTOS DE ALBERTO*

*En primer lugar quisiera dar gracias a Dios por permitirme llegar hasta a esta etapa tan importante de mi vida, por todas las bendiciones recibidas y por la maravillosa familia que me ha dado.*

*A mis padres José Carmen y Teresa les agradezco todo su infinito amor y esfuerzo para poder alcanzar esta meta juntos; los amare por siempre.*

*A mis hermanos Gonzalo, Yancy y Juan con quienes he pasado tantas cosas hermosas y aprendido a ser mejor persona, por todo su cariño y amor, siempre están en mi mente y corazón.*

*A Nelly, Lupita y Damiancito por ser ahora parte de nuestra familia gracias por estar con nosotros y que sepan que los amamos.*

*A todas mis familiares: tios, primos, sobrinos gracias por su cariño y amistad.*

*A Gaby por todo el cariño que me has dado y por tu gran apoyo. Te amo.*

*A mi gran amigo Israel y a su bella familia por el esfuerzo realizado en esta aventura de la tesis.*

*A mis amigos David y Arturo de toda la vida, a los de la facultad y del trabajo por su amistad incondicional.*

*A todos los integrantes de CROFI (Club de Robótica de la Facultad de Ingeniería) y a los profesores Dr. Jesús Savage, Ing. Agustín Millán y Rubén Anaya que siempre nos han apoyado en los diferentes proyectos y en especial a mi amigo Francisco Gálvez por su ejemplo de tenacidad y creatividad.*

*A mí querida Universidad Nacional Autónoma de México donde obtuve las mejores herramientas para vivir.*

*A mis profesores de la facultad por su ejemplo y valiosas enseñanzas.*

*A mi profesor y amigo Alejandro Mancilla Rosales por aceptar ser nuestro asesor en esta tesis y por guiarnos en la investigación de la misma con su gran capacidad y entusiasmo.*

*A nuestros sinodales por su apoyo para las mejoras de este proyecto de tesis.*

*En fin a todas las personas que han influido en mi vida muchas gracias.*

## *AGRADECIMIENTOS DE ISRAEL*

*A Dios por haberme guiado en todos los aspectos de la vida.*

*A mis padres Gabriel y Margarita por haberme apoyado para lograr este sueño. Sin ellos no hubiera sido posible. Los quiero mucho.*

*A mis hermanos Asur y Paola por apoyarme en los momentos más difíciles.*

*A mis abuelos Esperanza, Saturnino, María de Jesús y David, a todos mis tíos y a mis primos, en especial a Héctor y a Mario por haberme apoyado en mis estudios y en todas mis metas.*

*A todos mis amigos de la Facultad de Ingeniería por haber confiado en mí y haber pasado gratos momentos juntos.*

*A los chicos del coro y de la Prepa 5, por haberme hecho pasar los mejores momentos de mi vida con ustedes, en especial a Gabriel Vázquez Castro, Rosario Estrada Cruz, Beatriz García Vera Francisco Moreno Escobar, Miguel Gerardo Sandoval Chúa, Elia Escobar Sánchez, Erick Calva Fuentes, María Eugenia Pérez Rodríguez, Luis Alberto Heredia Cortés, Alma Téllez Romero, Marco Antonio Ramírez Rivera, Pedro González Gómez, Samuel González Gatica, Benjamín y Rosendo.*

*A todos mis amigos del trabajo.*

*En general, agradezco a todas las personas que me han ofrecido su amistad y su apoyo en distintos periodos de mi vida y en distintos lugares. Hacer una lista de ustedes sería interminable, pero siempre los llevo en mi corazón.*

*A Alberto Balderas por haberme aguantado todo este tiempo y ser un gran amigo.*

*A mis maestros de la Facultad por haberme dado los conocimientos para poder desempeñarme profesionalmente.*

*A la grandiosa Facultad de Ingeniería por ser la mejor del país.*

*A mi asesor Alejandro Mancilla por haberme guiado durante el presente trabajo con su paciencia y sus conocimientos.*

# ÍNDICE

**Página**

# INTRODUCCIÓN

Indudablemente en la elaboración de algún sistema de cómputo ya sea de tipo Web o de simple consola es necesario el contar con un acceso a datos de forma clara, rápida y eficiente, pues es quizás la parte medular de cualquier aplicación en la mayoría de las organizaciones.

Al contar con la experiencia de personas que diariamente interactúan con algún manejador de bases de datos y con determinado lenguaje de programación, se presentan de manera continua situaciones que hacen que la elaboración de código sea de manera rutinaria, tediosa y siempre con errores de conversión de tipos de datos, y más aún si hay que utilizar otra base de datos el trabajo se incrementa, pues muchos proyectos tienen embebido código SQL en la capa de presentación o en la de negocios.

Gracias a los patrones de diseño que dan solución a problemas de arquitectura de los sistemas, se da un primer paso para resolver esto, pues dividen en capas el acceso a datos; ya que en el nivel de la presentación y en las clase de modelo no tiene que existir alguna dependencia directa con las sentencias SQL que necesite la aplicación.

Con el patrón de diseño Model View Controller hay una separación de capas que permite un bajo acoplamiento entre ellas pero todavía se tienen que generar varias clases representativas del patrón de diseño DAO, además de los patrones Factory y Abstract Factory, por lo que la tarea de generación es aún de forma manual y requiere un esfuerzo extra por parte del programador.

De lo anterior nace la propuesta de tesis, pues se busco dar solución a la fabricación de código mediante alguna estrategia y gracias a las APIs de java para XML que son el corazón de la parte operativa del proceso de transformación se determinó que era viable abordar el problema y de una manera relativamente sencilla .

Java al contar con el API JDBC permite obtener provecho de sus características para la extracción de la información representativa de toda una base de datos relacional y de cada una de sus tablas.

Para la construcción de dicha herramienta también se deben entender conceptos de bases de datos, manipulación de archivos XML, patrones de diseño, construcción de interfaces gráficas de usuario en Swing entre otros.

Plasmando nuestras ideas, el planteamiento general consiste en construir una herramienta con un entorno gráfico para generar clases Java de manera automática realizando todo un mapeo de la información de una base de datos relacional (llaves primarias, llaves foráneas, tablas, campos, etc.) y que una vez compiladas podrán ser utilizadas ya sea por páginas JSPs o algunas Interfaces Gráficas de Usuario (GUI) (swing, awt, applets) para acceder y manipular distintas bases de datos relacionales realizando operaciones de inserción, borrado, actualización y búsqueda de registros.

Se propone apegarse a la metodología XP, además de utilizar elementos de RUP para tratar de obtener buenos resultados pues es una forma ágil de trabajar a fin de tener la disciplina necesaria para este proyecto.

# CAPÍTULO 1

# INGENIERÍA DE SOFTWARE DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

**Introducción**

En este capítulo se presentan los conceptos teóricos para comprender esta tesis, pues es importante entender definiciones de POO (programación orientada a objetos) por sus ventajas respecto a otros paradigmas de programación así como las diferentes metodologías para establecer con cual de ellas se debería trabajar y extraer los puntos que nos fuesen útiles.

También se presentan las herramientas de desarrollo que se utilizan para alcanzar el objetivo perseguido, como editores y generadores de código automático, por lo que se da un acercamiento a los entornos de desarrollo integrado y a las herramientas CASE y RAD para clasificar nuestro trabajo entre alguna de éstas dos últimas.

De igual forma se estudian los patrones de diseño que son la base fundamental de la presente tesis, ya que el código generado fue estructurado con el patrón DAO (Data Access Object) que nos permite un acceso más transparente de los clientes hacia alguna base de datos.

Para conocer un poco más lo que es el lenguaje de programación Java también se da una visión general de sus alcances y potencialidades, pues es muy versátil con respecto a otros, su gran fuerza radica en la alta disponibilidad y variedad de clases para tareas muy diversas, además de su independencia de la plataforma utilizada.

Por último se aborda la relación del estándar XML con Java, debido a que con algunas APIs se pueden intercambiar y procesar documentos tipo XML, y al desarrollar el proyecto la generación de código se realizó basándose en los procesos entre XML y Java.

## 1.1.- Metodologías

Las metodologías imponen una disciplina en el desarrollo de software con el fin de hacerlo más predecible y eficiente. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar inspirado por otras disciplinas de la ingeniería.

Las metodologías de software se han usado durante mucho tiempo. Y no han sido precisamente muy exitosas ni populares. La crítica más frecuente a estas metodologías es que son demasiado burocráticas. Hay mucho que hacer para seguir la metodología y entonces el ritmo entero del desarrollo se retrasa.

### 1.1.1.- Metodologías pesadas

#### 1.1.1.1.-RUP (Rational Unified Process, Proceso Unificado de Rational)

**Características del proceso**

- Es un proceso iterativo.
- Las actividades del RUP destacan en la creación y mantenimiento de modelos (especialmente aquellos especificados mediante UML) más que documentos en papel.
- El desarrollo bajo RUP está centrado en la arquitectura de software que guía el desarrollo del sistema.
- Soporta técnicas orientadas a objetos.
- Es un proceso configurable.
- Las actividades de desarrollo de RUP están dirigidas por los casos de uso.
- Fases e iteraciones

**Fases del RUP**

Una **fase** es un intervalo de tiempo entre dos metas importantes del proceso durante el cual se cumple un conjunto bien definido de objetivos, se completan artefactos y se toman las decisiones de pasar a la siguiente fase. RUP consta de las cuatro fases siguientes:

- Iniciación: Establecer la planificación del proyecto.
- Elaboración: Establecer un plan para el proyecto y una arquitectura completa.
- Construcción: Desarrollar el sistema.
- Transición: Proporcionar el sistema a sus usuarios finales.

Cada fase de RUP puede descomponerse en iteraciones. Una **iteración** es un ciclo completo de desarrollo que produce una versión de un producto ejecutable, que constituye un subconjunto del producto final en desarrollo, que luego se irá incrementando de iteración en iteración en el producto final.

El paso a través de las cuatro fases principales constituye un ciclo de desarrollo, y produce una generación de software. La primera pasada a través de las cuatro fases se denomina ciclo de desarrollo inicial.

**Flujos de trabajo de proceso**

- Modelado del negocio: Describe la estructura y la dinámica de la organización.
- Requisitos: Describe el método basado en casos de uso para extraer los requisitos.
- Análisis y diseño: Describe las diferentes vistas arquitectónicas.
- Implementación: Tiene en cuenta el desarrollo de software, la prueba de unidades y la integración.
- Pruebas: Describe los casos de pruebas, los procedimientos y las métricas para evaluación de defectos.
- Despliegue: Cubre la configuración del sistema entregable.
- Gestión de configuraciones: Controla los cambios y mantiene la integridad de los artefactos de un proyecto.
- Gestión de proyecto: Describe varias estrategias de trabajo en un proceso iterativo.
- Entorno: Cubre la infraestructura necesaria para desarrollar un sistema.

**Modelos**

Un artefacto es algún documento, informe o ejecutable que se produce, se manipula o se consume. Los modelos son el tipo de artefacto más importante en el RUP. Un modelo es una simplificación de la realidad, creada para comprender mejor el sistema que se está creando. En RUP tenemos nueve modelos:

- Modelo del negocio: Establece una abstracción de la organización.
- Modelo del domino: Establece el contexto del sistema.
- Modelo de casos de uso: Establece requisitos funcionales del sistema.

- Modelo de análisis (opcional): Establece un diseño de las ideas.
- Modelo de diseño: Establece el vocabulario del problema y su solución.
- Modelo del proceso (opcional): Establece los mecanismos de concurrencia y sincronización del sistema.
- Modelo de despliegue: Establece la topología hardware sobre la cual se ejecutará el sistema.
- Modelo de implementación: Establece las partes que se utilizarán para ensamblar y hacer disponible el sistema físico.
- Modelo de pruebas: Establece las formas de validar y verificar el sistema.

[Booch]

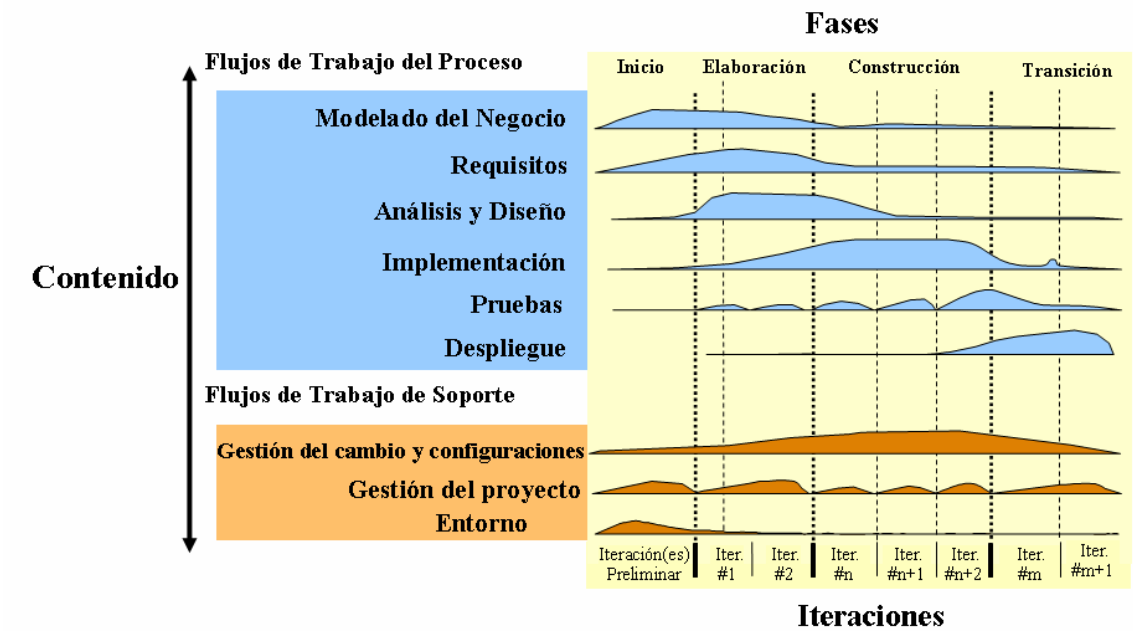



Fig. 1.1.- Diagrama de fases e iteraciones de RUP

### 1.1.2.- Metodologías ágiles

Durante algún tiempo se conocían como metodologías ligeras, pero el término aceptado ahora es metodologías ágiles. Para mucha gente la ventaja de estas metodologías ágiles es su reacción ante la burocracia de las metodologías pesadas. Estos nuevos métodos buscan un justo medio entre ningún proceso y demasiado proceso, proporcionando simplemente suficiente proceso para que el esfuerzo valga la pena.

La diferencia inmediata es que son menos orientados al documento, exigiendo una cantidad más pequeña de documentación para una tarea dada. De muchas maneras son más bien orientados al código: siguiendo un camino que dice que la parte importante de la documentación es el código fuente.

#### 1.1.2.1.-XP (Programación extrema)

En la programación extrema se da por supuesto que es imposible prever todo antes de empezar a codificar. Es imposible capturar todos los requisitos del sistema, saber qué es todo lo que tiene que hacer, y que no es posible hacer un diseño correcto al principio.

La idea de la programación extrema consiste en trabajar estrechamente con el cliente, haciéndole mini-versiones con mucha frecuencia (cada dos semanas). En cada mini-versión se debe hacer el mínimo de código y lo más simple posible para que funcione correctamente.

El cliente junto al equipo de desarrollo definen qué es lo que se quiere hacer. Para ello utilizan las "historias de usuario". Una historia de usuario es un texto de una o dos frases en las que se dice algo que debe hacer el sistema. Es más extensa que un requisito (que suele ser una frase corta) y menos que un caso de uso (que puede ser de una o dos páginas).

La programación extrema se basa en trece prácticas básicas que mencionan a continuación:

- Equipo completo: Forman parte del equipo todas las personas que tienen algo que ver con el proyecto, incluido el cliente y el responsable del proyecto.
- Planificación: Se hacen las historias de usuario y se planifica en qué orden se van a hacer y las mini-versiones. La planificación se revisa continuamente.
- Pruebas del cliente: El cliente, con la ayuda de los desarrolladores, propone sus propias pruebas para validar las mini-versiones.
- Versiones pequeñas: Las mini-versiones deben ser lo suficientemente pequeñas como para poder hacer una en pocas semanas. Deben ser versiones que ofrezcan algo útil al usuario final y no trozos de código que no pueda ver funcionando.
- Diseño simple: Hacer siempre lo mínimo imprescindible de la forma más sencilla posible. Mantener siempre sencillo el código.
- Pareja de programadores: Los programadores trabajan por parejas (dos delante de la misma computadora) y se intercambian las parejas con frecuencia (un cambio diario).
- Desarrollo guiado por las pruebas automáticas: Se deben realizar programas de prueba automática y deben ejecutarse con mucha frecuencia. Cuantas más pruebas se hagan, mejor.
- Mejora del diseño: Mientras se codifica, debe mejorarse el código ya hecho y que sea susceptible de ser mejorado. Extraer funcionalidades comunes, eliminar líneas de código innecesarias, etc.
- Integración continua: Debe tenerse siempre un ejecutable del proyecto que funcione y en cuanto se tenga una nueva pequeña funcionalidad, debe recompilarse y probarse.
- El código es de todos: Cualquiera puede y debe tocar y conocer cualquier parte del código. Para eso se hacen las pruebas automáticas.
- Normas de codificación: Debe haber un estilo común de codificación (no importa cual), de forma que parezca que ha sido realizado por una única persona.
- Metáforas: Hay que buscar frases o nombres que definan cómo funcionan las distintas partes del programa, de forma que sólo con los nombres se pueda dar una idea de qué hace cada parte del programa.

- Ritmo sostenible: Se debe trabajar a un ritmo que se pueda mantener indefinidamente. Esto quiere decir que no debe haber días muertos en que no se sabe qué hacer y que no se deben hacer un exceso de horas otros días. [WebXP].

### 1.2.- Ciclo de vida de un proyecto

En el nivel más simple, los proyectos tienen dos fases: planeación y ejecución. La planeación y la ejecución están bien para un proyecto simple y de poca duración. Para los proyectos grandes y de larga duración es necesario añadir otra capa al ciclo de vida del proyecto. Este puede ser ejecutado dividiendo cada fase: planear y ejecutar dentro de dos fases adicionales, dirigiendo hacia un ciclo de vida de análisis, diseño, desarrollo e implantación.

La tabla siguiente resume las fases de un proyecto de vida clásico y menciona las actividades que deben ser planeadas y ejecutadas en cada fase. [Allen]

| **Fase Primaria** | **Subfase** | **Actividades** |
|---|---|---|
| Análisis | Análisis de requerimientos. | Elaborar una relación de conceptos y definir detalladamente requerimientos y características externamente visibles del sistema. Escribir un plan de validación que trace la especificación de requerimientos. Dicho de otra forma: ¿Es posible realizar los requerimientos? |
| | Análisis del contexto del sistema. | Definir el contexto del sistema vía casos de uso y escenarios. Se definen mensajes externos, eventos y acciones. El sistema es tratado como una caja negra. Se puede proceder con cuidado y usar modelos del escenario. Para sistemas en tiempo real, se deben caracterizar la secuencia y detalles de sincronización de mensajes y respuestas. Dicho de otra forma: ¿Cómo podría verse la solución desde una perspectiva amplia? |
| | Análisis del modelo | Identificar las clases, objetos y asociaciones que resuelven el problema, utilizando diagramas de objetos y clases. El comportamiento de las respuestas es modelado por medio de diagramas de estado. La interacción entre objetos es mostrada con diagramas de secuencia y de colaboración. Dicho de otra forma: Un amplio refinamiento de la perspectiva de solución se logra descomponiendo subsistemas en clases de alto nivel. |
| Diseño | Diseño de la arquitectura. | Definir decisiones arquitecturales de importancia. La arquitectura física del sistema es modelada usando diagramas de distribución, la arquitectura de los componentes de software es modelada con diagramas de componentes y los modelos de concurrencia son usados en los diagramas de clases, los cuales muestran los objetos activos. Los patrones de diseño también son utilizados. Nota: Un elemento clave del diseño es que las dependencias fuertes con algún hardware específico, software o algún otro tipo de infraestructura son |

| | | |
|---|---|---|
| | | más habituales cuando más nos acercamos a la implantación. Por ejemplo, un arquitecto podría decidir usar BEA WebLogic como servidor J2EE. Un diseñador podría encontrar que, mientras intenta realizar el análisis gramátical de componentes XML podría decidir entre usar APIs de BEA o APIs JAXP. |
| | Diseño mecánico | Define la forma de colaboración de clases y objetos. Esta información es plasmada en diagramas de clase y objetos. Los diagramas de secuencia y colaboración plasman instancias específicas de colaboración y los diagramas de estado nos terminan de mostrar el comportamiento total. |
| | Diseño detallado | Define el comportamiento detallado y la estructura individual de las clases por medio de diagramas de actividad y notas. |
| Desarrollo | | Desarrolla la codificación de las clases, la definición de la base de datos y mensajes estructurados en lenguaje objetivo, el manejador de base de datos y el sistema de comunicación. |
| Implantación | Pruebas unitarias | Prueba la estructura interna y el comportamiento de cada clase. |
| | Pruebas de integración | Prueba la integración de varios componentes. Esto se hace recursivamente en múltiples niveles de descomposición basados en la escala del sistema. |
| | Pruebas de validación | Prueba el sistema liberado con los requerimientos que se definieron en el plan de pruebas. |
| | Sistema liberado | Entregar el sistema liberado, la guía de usuario y demás documentación operacional al usuario y al personal de soporte técnico. |

### 1.3 Herramientas de desarrollo

Debido a que la herramienta que se propone para la presente tesis facilita en gran medida la generación de código, debemos clasificarla de alguna manera, para poder ubicarla en algún contexto determinado y saber como involucrarla con otras herramientas que puedan hacer uso de ella.

#### 1.3.1.- Herramientas CASE.

La tecnología conocida con el nombre de CASE (Computer Aided/Assisted Software/System Engineering), supone la "informatización de la informática", es decir "la automatización del desarrollo del software", contribuyendo así a elevar la productividad y la calidad en el desarrollo de sistemas de información.

Este nuevo enfoque a la hora de construir software, persigue los siguientes objetivos:

- Permitir la aplicación práctica de metodologías estructuradas, lo que resulta muy difícil sin emplear herramientas.

- Mejorar la calidad del software.

- Facilitar la realización de prototipos, y el desarrollo conjunto de aplicaciones.
- Simplificar el mantenimiento de los programas.
- Mantener un estándar en la documentación.
- Aumentar la portabilidad de las aplicaciones.
- Facilitar la reutilización de componentes software.
- Permitir un desarrollo visual de las aplicaciones, mediante la utilización de gráficos.

De una manera muy esquemática, se puede afirmar que una herramienta CASE se compone de los siguientes elementos:

- Repositorio (diccionario) donde se almacenan los elementos definidos o creados por la herramienta, y que se basa en un manejador de base de datos o en un sistema de administración de archivos.
- Metamodelo (no siempre visible), que define las técnicas y metodologías soportadas por la herramienta, y que es conveniente que pueda ser extensible por parte del usuario.
- Generador de informes, que permite obtener toda la documentación que describe el sistema de información desarrollado; documentación que está asociada a las técnicas y metodologías.
- Herramienta de carga/descarga de datos, que permite cargar el repositorio de la herramienta CASE con datos provenientes de otros sistemas, o generar a partir de la propia herramienta esquemas de bases de datos, programas, etc.
- Interfaz de usuario, que constará de editores de texto y herramientas de diseño gráficos, que permitan mediante la utilización de un sistema de ventanas, iconos y menús, con la ayuda del ratón, definir los diagramas, matrices, etc. que incluyen las distintas metodologías. Lo que se conoce usualmente por las siglas inglesas WIMP (Windows, Icons, Mouse y menús Pull-down).

Comprobación de errores, facilidades que permiten llevar a cabo un análisis de la exactitud, integridad y consistencia de los esquemas generados por la herramienta. [WebCase]

#### 1.3.1.1 Categorías de herramientas CASE

A continuación listamos las categorías de herramientas CASE más frecuentes.

- Herramientas de análisis y diseño
- Herramientas de generación de código y documentación
- Herramientas de prueba
- Herramientas de administración de configuración
- Herramientas de ingeniería inversa

### 1.3.2 Herramientas RAD.

Los actuales entornos RAD (Rapid Application Development) permiten no sólo el desarrollo de aplicaciones finales sino también el diseño de prototipos eficientes, y ofrecen:

- Entorno de desarrollo visual, que incluye, entre otros, el menú propio del entorno, la paleta de componentes, la ventana de propiedades (que refleja las propiedades asociadas al control activo) y el editor de código.

- Controles de alto nivel, que suelen ser de dos tipos: los incorporados en la propia herramienta y aquellos externos que pueden ser añadidos. Estos controles se disponen en forma de iconos en una paleta de componentes.

- Acceso directo a las distintas partes del código asociado a cada uno de los objetos, y que se encuentra en la ventana del editor de código.

Las aplicaciones diseñadas se basan en entornos gráficos de usuario (GUI) fáciles de manejar y con el siguiente proceso de desarrollo:

- Creación de un formulario vacío que contendrá los componentes del interfaz de la aplicación (controles).

- Selección de los componentes apropiados del conjunto de los disponibles en la paleta de componentes en forma de iconos. Estos pueden luego ser redimensionados una vez colocados en el formulario.

- Particularización de las propiedades de los controles en base a los requisitos de la aplicación.

- Escritura de código para los distintos eventos significativos asociados al control.

- Ejecuciones de prueba dentro del entorno de desarrollo.

- Modificación de la aplicación durante el proceso de desarrollo hasta la generación de un archivo ejecutable (.exe o .jar) definitivo.

- Utilización de herramientas de apoyo para depurar y refinar el código.

También las herramientas RAD permiten crear aplicaciones con bases de datos. La conexión con una base de datos se puede efectuar de dos formas:

- Directamente, utilizando el motor de dicha base de datos.

A través de enlaces remotos (vía ODBC o JDBC). [WebRAD]

### 1.3.3.- Entornos de desarrollo integrado

Un entorno integrado del desarrollo (IDE) es un ambiente de programación que se ha conformado como un programa de aplicación, consistiendo típicamente en un editor de texto, de un compilador, de un programa de depuración, y de un constructor de la interfaz gráfica de usuario (GUI).

Los IDEs proporcionan un marco de trabajo (framework) fácil de manejar para muchos lenguajes de programación modernos, Basic (Visual Basic), Java (NetBeans, JCreator, WebSphere, Eclipse, etc.) y PowerBuilder. [WebIDE]

#### 1.3.3.1 NetBeans

El IDE NetBeans nos permite principalmente:

- Desarrollar y mantener proyectos.

- Crear y editar código fuente Java.

- Compilar archivos con extensión Java.
- Depurar programas.
- Empaquetar y desplegar aplicaciones.
- Documentar proyectos.

#### 1.3.3.2.- XML-SPY

XMLSpy es una herramienta de soporte para la construcción de multitud de componentes XML.

Entre sus características principales están:

- Nos permite editar archivos XML, XSL, HTML, DTDs entre otras más.
- Checa que un documento esté bien formado y que sea válido también.
- Permite visualizar archivos resultado de transformar archivos XML con plantillas XSL.

Estos IDEs fueron los que más se utilizan para el desarrollo de esta tesis pues prácticamente contienen la mayoría de los elementos necesarios para el desarrollo de código.

## 1.4.- Conceptos y principios de la programación orientada a objetos

### 1.4.1.- Programación orientada a objetos

La programación orientada a objetos es un modelo de programación que utiliza objetos, ligados mediante mensajes para la solución de problemas. Puede considerarse como una extensión natural de la programación estructurada en un intento de potenciar los conceptos de modularidad y reutilización del código.

### 1.4.2.- Mecanismos básicos de la programación orientada a objetos

#### 1.4.2.1.- Objetos

Un programa orientado a objetos se compone solamente de objetos, entendiendo por objeto un encapsulamiento genérico de datos y de los métodos para manipularlos. Dicho de otra forma, un objeto es una entidad que tiene unos atributos particulares, las propiedades, y unas formas de operar sobre ellos, los métodos.

Por ejemplo, una ventana de una aplicación Windows es un objeto. El color de fondo, la anchura, la altura, etc. son propiedades. Las rutinas, lógicamente transparentes al usuario, que permiten maximizar la ventana, minimizarla, etc. son los métodos.

#### 1.4.2.2.- Mensajes

Cuando se ejecuta un programa orientado a objetos, los objetos están recibiendo, interpretando y respondiendo a mensajes de otros objetos. Esto marca una clara diferencia con respecto a los elementos de datos pasivos de los sistemas tradicionales. En la POO un mensaje es asociado con un método, de tal forma que cuando un objeto recibe un mensaje la respuesta a ese mensaje es ejecutar el método asociado.

#### 1.4.2.3.- Métodos

Un método se implementa en una clase de objetos y determina cómo tiene que actuar el objeto cuando recibe el mensaje vinculado con ese método. A su vez, un método puede también enviar mensajes a otros objetos solicitando una acción o información.

Adicionalmente las propiedades (atributos) definidas en la clase permitirán almacenar información para dicho objeto.
Cuando se diseña una clase de objetos, la estructura más interna del objeto se oculta a los usuarios que lo vayan a utilizar, manteniendo como única conexión con el exterior, los mensajes. Esto es, los datos que están dentro de un objeto solamente podrán ser manipulados por los métodos asociados al propio objeto.

Según lo expuesto, podemos decir que la ejecución de un programa orientado a objetos realiza fundamentalmente tres cosas:

- Crea los objetos necesarios.
- Los mensajes enviados entre los objetos dan lugar a que se procese internamente la información.
- Finalmente, cuando los objetos no son necesarios, son borrados, liberándose la memoria ocupada por los mismos.

#### 1.4.2.4.- Clases

Una clase es un tipo de objetos definido por el usuario. Una clase equivale a la generalización de un tipo específico de objetos.

Un objeto de una determinada clase se crea en el momento en que se define una variable de dicha clase. Por ejemplo, la siguiente línea declara el objeto cliente de la clase o tipo Cuenta.

```
Cuenta cliente = new Cuenta(); //nueva cuenta
```

Cuando se escribe un programa utilizando un lenguaje orientado a objetos, no se definen objetos verdaderos, se definen clases de objetos, donde una clase se ve como una plantilla para múltiples objetos con características similares.

### 1.4.3.- Características de la programación orientada a objetos

#### 1.4.3.1.- Abstracción

Por medio de la abstracción conseguimos generalizar y centrarnos en los aspectos que permitan tener una visión global del problema sin detenernos en los detalles concretos de las cosas que no interesen en cada momento.

#### 1.4.3.2.- Encapsulamiento

Esta característica permite ver a un objeto como una caja negra en la que se ha introducido de alguna manera toda la información relacionada con dicho objeto. Esto permitirá manipular los objetos como unidades básicas, permaneciendo oculta su estructura interna.

La abstracción y el encapsulamiento están representadas por la clase. La clase es una abstracción, porque en ella se definen las propiedades o atributos de un determinado conjunto de objetos con características comunes, y es un encapsulamiento porque constituye una caja negra que encierra tanto los datos que almacena cada objeto como los métodos que permiten manipularlos.

#### 1.4.3.3.- Herencia

La herencia permite el acceso automático a la información contenida en otras clases. De esta forma, la reutilización del código esta garantizada. Con la herencia todas las clases están clasificadas en una jerarquía estricta. Cada clase tiene su superclase (la clase superior a la jerarquía), y cada clase puede tener una o más subclases (las clases inferiores en la jerarquía).

Las clases que están en la parte inferior en la jerarquía se dice que heredan de las clases que están en la parte superior en la jerarquía.

El término heredar significa que las subclases disponen de todos los métodos y propiedades de su superclase. Este mecanismo proporciona una forma rápida y cómoda de extender la funcionalidad de una clase.

#### 1.4.3.4.- Polimorfismo

Esta característica permite implementar de múltiples formas un mismo método, dependiendo cada una de ellas de la clase sobre la que se realice la implementación. Esto hace que se pueda acceder a una variedad de métodos distintos (todos con el mismo nombre) utilizando exactamente el mismo medio de acceso. [Ceballos]

### 1.5.- Patrones de diseño

Los patrones de diseño o patrones son soluciones a problemas que se repiten en un contexto dado. Intentan reunir y documentar la solución principal a un problema dado. Estos son identificados y documentados en una forma que sea fácil de compartir, de discutir, y de entender.

#### 1.5.2.- Ventajas de usar patrones de diseño

Los patrones del diseño son benéficos porque describen un problema que ocurre en varias ocasiones, y entonces explican la solución al problema de una manera que se pueda utilizar demasiadas veces.

Los patrones del diseño son útiles por las razones siguientes:

- Ayudan a diseñadores a centrarse rápidamente en soluciones si los diseñadores pueden reconocer los patrones que han sido exitosos en el pasado.
- El estudio de los patrones puede inspirar a los diseñadores a presentar nuevas y únicas ideas.
- Proporcionan un lenguaje común para las discusiones del diseño.
- Proporcionan soluciones a los problemas del mundo real.
- Su formato captura el conocimiento y documenta las mejores prácticas para un campo de acción.
- Documentan las decisiones y el análisis razonado que conducen a la solución.
- Reutilizan la experiencia de los antecesores.
- Comunican el discernimiento ganado previamente.
- Describen las circunstancias (cuándo y dónde), las influencias (de quiénes y de qué), y la resolución (cómo y porqué equilibra las influencias) de una solución.

Algunas características comunes de los patrones son:

- Se observan a través de la experiencia.
- Se escriben típicamente en un formato estructurado.
- Evitan la reinvención de la rueda.

- Existen en diversos niveles de abstracción.
- Experimentan mejoras continuas.
- Son artefactos reutilizables.
- Comunican diseños y las mejores prácticas.
- Se pueden utilizar juntos para solucionar un problema más grande.

Sin embargo, los patrones no lo son todo ni el fin de todo, no son de ninguna manera una panacea, y no pueden ser aplicados universalmente a todas las situaciones. [Deepak]

### 1.5.3.- Patrones de diseño de la "Pandilla de los Cuatro" (GoF).

#### 1.5.3.1.- Patrones de creación

Los patrones de creación se refieren a la manera en que se crean los objetos. Se utilizan estos patrones cuando una decisión debe ser tomada al tiempo de instanciar una clase. Típicamente, los detalles de la clase concreta que será instanciada son ocultados desde (y desconocidos para) la llamada a la clase a través de una clase abstracta que sabe solamente sobre la clase abstracta o la interfaz que la pone en ejecución. Los siguientes patrones de creación son descritos por el GoF. [Allen]

| Nombre del patrón | Descripción |
|---|---|
| Abstract Factory | Proporciona una interfaz para crear familias de objetos dependientes o relacionados sin especificar las clases concretas. |
| Builder | Separa la construcción de un objeto complejo de su representación, de esta forma el proceso de construcción puede crear distintas representaciones. |
| Factory Method | Define una interfaz para la creación de un objeto, permitiendo a las subclases decidir cuándo una clase será instanciada. Permite a una clase delegar la actual instanciación a las subclases. |
| Prototype | Especifica el tipo de objetos que se crearán utilizando una instancia prototípica y crea nuevos objetos por medio de la copia de este prototipo. |
| Singleton | Asegura que una clase pueda tener sólo una instancia. |

#### 1.5.3.2.- Patrones estructurales

Los patrones estructurales se refieren a la composición o a la organización de las clases y de los objetos, cómo heredan las clases, y cómo se componen de otras clases. Los patrones estructurales comunes incluyen los patrones Adapter, Proxy y Decorator. Estos patrones son similares a los que introducen un nivel de relación indirecta entre una clase del cliente y una clase que desea utilizar. Sin embargo, sus propósitos son diferentes. Adapter utiliza la relación indirecta para modificar la interfaz de una clase para hacerla más fácil de utilizar por el cliente. Decorador utiliza la relación indirecta para agregar comportamiento a una clase, sin afectar la clase del cliente. Proxy utiliza la relación indirecta de forma transparente para proporcionar un suplente para otra clase. Los patrones estructurales siguientes son descritos por el GoF. [Allen]

| Nombre del patrón | Descripción |
|---|---|
| Adapter | Convierte la interfaz de una clase en otra interfaz que espera el cliente. Permite a las clases trabajar juntas en una situación que de otra forma no podría suceder por las interfaces incompatibles. |
| Bridge | Desacopla la abstracción de su implantación, de forma que ambas cambian de forma independiente. |
| Composite | Compone objetos dentro de estructuras de árbol que representan jerarquías parciales. Permite a los clientes |

| | |
|---|---|
| | considerar objetos de forma individual y composiciones de objetos de manera uniforme. |
| Decorador | Agrega dinámicamente responsabilidades a un objeto. Proporciona una alternativa flexible a la subclasificación al momento de ampliar su funcionalidad. |
| Facade | Proporciona una interfaz unificada a un conjunto de interfaces dentro de uno o más subsistemas. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de utilizar. |
| Flyweight | Utiliza la distribución para soportar un gran número de objetos de una forma eficiente. |
| Proxy | Proporciona un contenedor o sustituto a un objeto para controlar el acceso a éste. |

### 1.5.3.3.- Patrones de comportamiento

Los patrones del comportamiento se refieren a la interacción y a la responsabilidad de los objetos. Ayudan a hacer un comportamiento complejo más manejable especificando las responsabilidades de objetos y las maneras en que se comunican. Los patrones del comportamiento siguientes son descritos por el GoF. [Allen]

| **Nombre del patrón** | **Descripción** |
|---|---|
| Chain of responsibility | Evita unir el emisor de una petición a su receptor por medio de dar oportunidad a más de un objeto de manejar la petición. La recepción de objetos es encadenada y se pasa la petición a lo largo de la cadena hasta que ésta es manejada. |
| Command | Encapsula una petición como un objeto, permitiendo al cliente ser parametrizado con diferentes peticiones, colas o registro de peticiones y ser capaz de soportar la operación de deshacer. |
| Interpreter | Dado un lenguaje, define una representación para su gramática junto con un intérprete de la gramática que utiliza la representación para interpretar oraciones del lenguaje. |
| Iterator | Proporciona una forma secuencial de acceder a los elementos de una colección de objetos sin hacerlo a través de su representación fundamental. |
| Mediator | Define un objeto que encapsula la forma en cómo interactúan un conjunto de objetos. Estimula el bajo acoplamiento, cuidando que los objetos hagan referencias directas entre ellos y cambiando su interacción de forma independiente. |
| Memento | Sin violar el encapsulamiento, captura y exterioriza el estado interno de un objeto, de forma que el estado esencial de un objeto puede ser restaurado posteriormente. |
| Observer | Define una dependencia uno a muchos entre objetos de forma que cuando un objeto cambia de estado, todas sus dependencias (suscriptores) son notificadas y actualizadas automáticamente. |
| State | Permite a un objeto cambiar su comportamiento cuando cambia su estado interno; parecerá como si el objeto cambiara su clase. |
| Strategy | Define una familia de algoritmos, encapsulando a cada uno y haciéndolos intercambiables. Permite que el algoritmo cambie independientemente de los clientes que lo usan. |
| Template method | Define el esqueleto de un algoritmo (función) en una operación, delegando algunos pasos a las subclases. Permite a las subclases redefinir ciertos pasos de un algoritmo sin cambiar la estructura del algoritmo. |
| Visitor | Representa una operación que será realizada a los elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos a través de |

| | |
|---|---|
| | los cuáles esta opera. |

### 1.5.4.- Patrones de diseño J2EE

Similar a los patrones de diseño del GoF, los patrones J2EE se analizan en varias secciones que se refieren a las capas siguientes: Presentación, negocio e integración.

#### 1.5.4.1.- Patrones de la capa de presentación

La capa de presentación encapsula la lógica requerida para atender a los clientes que tienen acceso a un sistema. Los patrones de la capa de presentación interceptan una petición del cliente y después proporcionan facilidades tales como el ingreso de un solo cliente por contraseña, administración de la sesión del cliente, y acceso a los servicios de la capa de negocio antes de construir y de entregar la respuesta de nuevo al cliente. [Deshmuk]

| **Nombre del patrón** | **Descripción** |
|---|---|
| Decorating Filter/Intercepting Filter | Es un objeto que está situado entre el cliente y los componentes web. Preprocesa una petición y postprocesa la respuesta. |
| Front Controller/Front Component | Es un objeto que acepta todas las peticiones del cliente y las envía o encamina hacia los manejadores apropiados. El patrón Front Controller puede dividir la funcionalidad anterior en dos diferentes objetos: el Controlador Frontal y el Despachador. En este caso el Controlador Frontal acepta todas las peticiones del cliente y hace la autentificación y el Despachador las envía o encamina hacia los manejadores apropiados. |
| View Helper | Es un objeto auxiliar que encapsula la lógica de acceso a los datos en beneficio de los componentes de presentación. Por ejemplo, los Java Beans pueden ser utilizados como patrones View Helper para páginas JSP. |
| Composite View | Es un objeto vista que se compone de la unión de otras vistas. Por ejemplo, una página JSP que incluye otras páginas JSP y HTML utilizando la directiva include o la acción include es un patrón Composite View. |
| Service to Worker | Es un tipo de Modelo-Vista-Controlador con el controlador actuando como Front Controller, pero con un punto importante: aquí el Despachador (el cual es una parte del Front Controller) utiliza View Helpers de gran alcance y añade un administrador de vistas. |
| Dispatcher View | Es un tipo de Modelo-Vista-Controlador con el controlador actuando como Front Controller, pero con un punto importante: aquí el Despachador (el cual es una parte del Front Controller) no utiliza View Helpers y hace muy poco por administrar las vistas. La administración de vistas el manejada por las propias vistas. |

#### 1.5.4.2.- Patrones de la capa de negocio

La capa de negocio proporciona los servicios requeridos por los clientes de la aplicación y contiene los datos y la lógica de negocio. Todo el procesamiento de negocio para la aplicación se recolecta y se pone en esta capa. Los Enterprise JavaBeans (EJB) son una de las maneras de poner el proceso de negocio en ejecución en esta capa. Estos son los patrones disponibles para la capa de negocio. [Deshmuk]

| **Nombre del patrón** | **Descripción** |
|---|---|
| Business Delegate | Es un objeto que reside en la capa de presentación y beneficia a otros componentes de la capa de presentación por medio de llamadas a métodos remotos de los objetos de la capa de |

| | |
|---|---|
| | negocio. |
| Composite Entity | Es un bean de entidad que modela, representa y administra un conjunto de objetos persistentes interrelacionados en vez de representarlos como beans de entidad individuales. Un bean Composite Entity representa un conjunto de objetos. |
| Value Object/Data Transfer Object/Replicate Object | Es un objeto serializable para transferir datos a través de la red. |
| Session Facade/ Session Entity Facade/Distributed Façade | Es un objeto que reside en la capa de negocios que actúa como un punto de entrada a dicha capa y administra el flujo de trabajo de los objetos de servicio de negocios, tales como beans de sesión, beans de entidad y Data Access Objects. El Session Facade en sí mismo es usualmente implantado como un bean de sesión. |
| Aggregate Entity | Es un objeto (bean de entidad) que se compone o está unido a otros beans de entidad. |
| Value Object Assembler | Es un objeto que reside en la capa de negocio y crea Value Objects tan pronto como se requieran. |
| Value List Handler/Page-by-Page Iterator/Paged List | Es un objeto que administra la ejecución de consultas y guarda y procesa los resultados. Es usualmente implantado como un bean de sesión, atendiendo una parte de los resultados obtenidos hacia el cliente cuando sea necesario. |
| Service Locator | Es un objeto que realiza la tarea de localizar servicios de negocio en beneficio de otros componentes de la capa. Generalmente se ubica en la capa de presentación y es utilizado por el Bussines Delegate para mejorar los objetos de servicio de negocios. |

#### 1.5.4.3.- Patrones de la capa de integración

Esta capa es responsable de tener acceso a recursos y a los sistemas externos, tales como almacenes de datos relacionales y no relacionales y cualquier aplicación heredada. Un objeto de la capa de negocio utiliza la capa de integración cuando requiere los datos o los servicios que residen en el nivel del recurso. Los componentes en esta capa pueden utilizar JDBC, la tecnología de conexión del J2EE, o algún otro software propietario para acceder a los datos en el nivel del recurso. Estos son los patrones disponibles para la capa de integración. [Deshmuk]

| **Nombre del patrón** | **Descripción** |
|---|---|
| Data Access Object | Es un objeto que habla con la base de datos y proporciona otros componentes de aplicación. Sirve como una simple, limpia y común interfaz para acceder a los datos y para reducir la dependencia de otros componentes en los detalles de la utilización de la base de datos. |
| Service Activator | Es un objeto auxiliar en el procesamiento asincrónico de métodos de negocio. |

## 1.6.- Lenguaje y plataforma Java

### 1.6.1.- Características del lenguaje Java

- Simple
- Orientado a objetos
- Distribuido
- Robusto

- Seguro
- De arquitectura neutral
- Portable
- Interpretado
- De alto rendimiento
- Multiproceso
- Dinámico

[Horstmann]

### 1.6.2.- Plataforma Java

#### 1.6.2.1.- ¿Qué es plataforma?

No existe una definición que plasme con exactitud el significado de este concepto, por lo cual se irá explicando poco a poco. Java es una plataforma en tres sentidos:

- Ofrece un potente lenguaje de programación
- Es independiente de la arquitectura hardware / software
- Ofrece gran cantidad de APIs estandarizadas para facilitar el desarrollo de software. [WebJava]

#### 1.6.2.2.- Java como lenguaje de programación

Java es un lenguaje de programación fuertemente orientado a objetos (se usan abstracciones de objetos siempre que se puede), sencillo de aprender y de usar, y con una gran flexibilidad. Entre sus fortalezas se encuentran una gran facilidad para realizar la administración de memoria (tarea simplificada gracias a la presencia del recolector de basura heredado de Smalltalk) y una mayor sencillez para el manejo de objetos que la que pueden presentar lenguajes como C++, ya que, entre otras cosas, en Java desaparece el concepto de apuntador (realmente no desaparece, sino que en Java sólo se usan apuntadores, de modo que se elimina la dificultad de discernir entre qué es un apuntador y qué no).

El lenguaje, sintácticamente, hereda de C++ gran parte de su aspecto, y de SmallTalk su fuerte orientación a objetos, pero fusionando ambos en un lenguaje muy potente y claro, con el que el paso de la fase de diseño de una aplicación a la de implementación se hace mucho más sencillo (razón para que sea uno de los lenguajes preferidos para programar aplicaciones creadas partiendo de unas necesarias fases de análisis y diseño, que desgraciadamente no siempre se llevan a cabo).

#### 1.6.2.3.- Java como arquitectura independiente

La mayoría de los lenguajes de programación están pensados de manera que al compilar un programa el resultado sea código nativo. Es decir, instrucciones máquina. Los sistemas operativos y gran parte del software que utilizamos cotidianamente están hechos de esta manera.

Sin embargo, esta forma de desarrollar software nos ata a dos cosas:

- La arquitectura hardware: ya que las instrucciones de máquina creadas son inteligibles por un determinado procesador, pero no lo serán por otros procesadores con juegos de instrucciones diferentes.

- El sistema operativo: ya que el programa creado necesitará comunicarse y hacer uso de las funciones que el sistema operativo le ofrece, hasta para algo tan sencillo como sacar un carácter por pantalla o, simplemente, terminar su ejecución.

Podemos esquematizar esta situación de la siguiente manera:


*Disposición de capas: aplicación nativa*

Fig. 1.2.- Diagrama de una aplicación nativa

Si una de las dos piezas (sistema operativo, arquitectura hardware) cambia, nuestro programa ya no será válido.

En contraposición a esto, Java nos ofrece el concepto de máquina virtual. Esto implica la creación de un procesador software (virtual) que se dedique a procesar las instrucciones del programa y se entienda con el hardware y el sistema operativo. Esquemáticamente:


*Disposición de capas: máquina virtual*

Fig. 1.3.- Diagrama de una aplicación sobre una máquina virtual

De esta manera hacemos que nuestra aplicación ya no tenga que entendérselas con el hardware ni con el sistema operativo, sino que tan sólo tenga que comunicar sus instrucciones a la máquina virtual.

Esto nos independiza del hardware y hace que sólo dependamos de la máquina virtual. Si creamos varias máquinas virtuales para distintas combinaciones de hardware/sistema operativo, teóricamente podremos ejecutar el mismo programa sin recompilar ni modificar en absoluto en todas las arquitecturas para las que hayamos creado una máquina virtual.

De ese modo, cuando creamos una aplicación Java en Linux y la compilamos lo que se genera no es código nativo sino código de bytes (código que entiende la máquina virtual), y sabemos que nuestra aplicación ya compilada podrá ser usada en Linux sobre Intel, Windows sobre Intel, Solaris sobre Intel y Sparc (de 32 y 64 bits) y en cualquier otra arquitectura para la que se haya desarrollado una máquina virtual Java.

#### 1.6.2.4.- Java como conglomerado de APIs

Con cada una de las versiones que Sun lanza del JDK, se acompaña de una serie de bibliotecas con clases estándar que valen como referencia para todos los programadores en Java.

Estas clases se pueden incluir en los programas Java, sin temor a errores de portabilidad. Además, están bien documentadas (mediante páginas Web), y organizadas en paquetes y en un gran árbol de herencia.

A este conjunto de paquetes (o bibliotecas) se le conoce como la API de Java (Application Programming Interface).

En este apartado explicaremos los paquetes básicos de la API de Java, aunque algunos de ellos tienen subpaquetes.

- Paquetes de utilidades

  java.lang: Fundamental para el lenguaje. Incluye clases como String o StringBuffer.

  java.io: Para la entrada y salida a través de flujos de datos, y archivos del sistema.

  java.util: Contiene colecciones de datos y clases, el modelo de eventos, facilidades horarias, generación aleatoria de números, y otras clases de utilidad.

  java.math: Clases para realizar aritmética con la precisión que se desee.

  java.text: Clases e interfaces para manejo de texto, fechas, números y mensajes de una manera independiente a los lenguajes naturales.

  java.security: Clases e interfaces para seguridad en Java: Encriptación RSA, etc.

- Paquetes para el desarrollo gráfico

  java.applet: Para crear applets y clases que las applets utilizan para comunicarse con su contexto.

  java.awt: Para crear interfaces con el usuario, y para dibujar imágenes y gráficos.

  javax.swing: Conjunto de componentes gráficos que funcionan igual en todas las plataformas que Java soporta.

  javax.accesibility: Da soporte a clases de accesibilidad para personas discapacitadas.

  java.beans: Para el desarrollo de JavaBeans.

- Paquetes para el desarrollo en red

  java.net: Clases para aplicaciones de red.

  java.sql: Paquete que contiene el JDBC, para conexión de programas Java con bases de datos.

java.rmi: Paquete RMI, para localizar objetos remotos, comunicarse con ellos e incluso enviar objetos como parámetros de un objeto a otro.

org.omg.CORBA: Facilita la posibilidad de utilizar OMG CORBA, para la conexión entre objetos distribuidos, aunque esté codificados en distintos lenguajes.

org.omb.CosNaming : Da servicio al IDL de Java, similar al RMI pero en CORBA

### 1.6.3.- Java 2 Standard Edition (J2SE)

Hay dos productos principales dentro plataforma Java 2 Standard Edition:

- Java Runtime Enviroment (JRE): Java RE proporciona las bibliotecas, la máquina virtual de Java, y otros componentes necesarios para que se ejecuten los applets y aplicaciones escritas en el lenguaje de programación Java. No contiene herramientas y utilerías tales como compiladores o depuradores para el desarrollo de applets y aplicaciones.

- Java 2 Software Development Kit (SDK): Java 2 SDK es un superconjunto de JRE. Java 2 SDK contiene todo el RE más herramientas adicionales tales como compiladores y depuradores que sean necesarios para el desarrollo de applets y aplicaciones.

Java RE es más pequeña y por lo tanto más fácil descargar o distribuir con el software que Java 2 SDK. La mayoría de los usuarios de la tecnología de Java son gente que solo desea que funcionen los applets y las aplicaciones desarrollados por otros. No están interesadas en desarrollar ninguna aplicación ellos mismos. La relativamente pequeña Java RE se hace para tales usuarios finales. Pueden descargar Java RE ellos mismos de Internet, o los vendedores de software pueden incluirla en sus propios productos. El usuario final típico no necesita la más abultada Java 2 SDK con sus herramientas de desarrollo. [DOCJDK142]

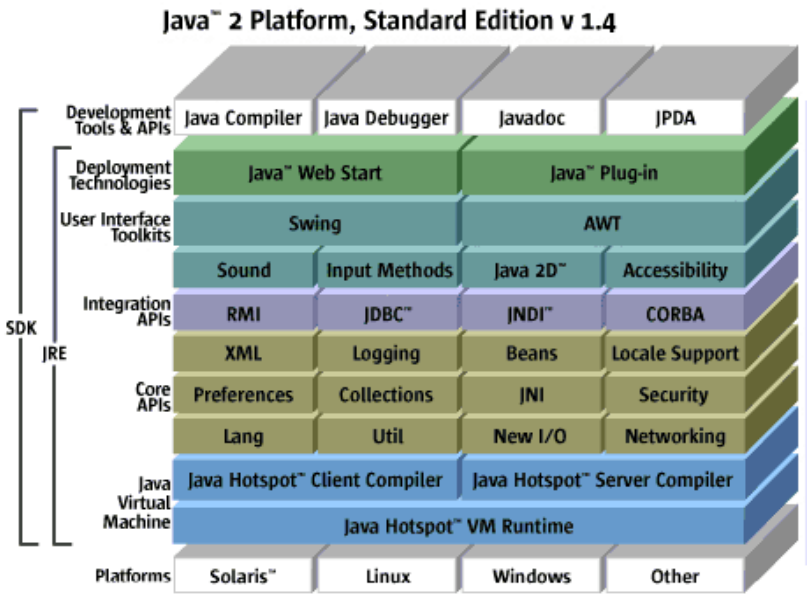



Fig. 1.4.- Diagrama de la plataforma J2SE 1.4

### 1.6.4.- Java2 Enterprise Edition (J2EE)

J2EE es una plataforma para desarrollar aplicaciones de software empresarial distribuido. Desde su inicio, el lenguaje Java, ha experimentado enorme adopción y crecimiento. Más y más tecnologías han llegado a ser parte de la plataforma Java, y nuevas APIs y estándares se han desarrollado para tratar varias necesidades.

Eventualmente, Sun y un grupo de líderes de industria, bajo auspicios del Proceso de la Comunidad de Java (JCP), unificó todos estos estándares empresariales y APIs dentro de la plataforma J2EE.

La arquitectura de J2EE consiste en las capas siguientes:

- Capa de cliente: La capa de cliente interactúa con el usuario y despliega información del sistema al usuario. La plataforma de J2EE soporta diversos tipos de clientes, incluyendo clientes HTML, applets, y aplicaciones Java.

- Capa web: La capa web genera la lógica de presentación y acepta respuestas del usuario a partir de los clientes de presentación, que son típicamente clientes HTML, Java applets, y otros clientes web. De acuerdo con la petición recibida del cliente, la capa de presentación genera la respuesta apropiada a una petición del cliente que este recibe. En la plataforma J2EE, servlets y JSPs en un contenedor web implantan esta capa.

- Capa de negocio: Esta capa maneja la lógica del negocio fundamental de la aplicación. La capa del negocio proporciona las interfaces necesarias a los componentes subyacentes de servicio de negocios. Los componentes de negocio están típicamente implantados como componentes EJB con el soporte de un contenedor de EJBs que facilita el ciclo de vida del componente y maneja la persistencia, transacciones y la asignación de recursos.

- Capa EIS: Esta capa es responsable de los sistemas de información empresariales, incluyendo sistemas de bases de datos, sistemas de procesamiento transaccional, sistemas de herencia y sistemas de planeamiento de recursos empresariales. La capa EIS es el punto donde las aplicaciones J2EE se integran con las que no son J2EE o con los sistemas de herencia.

El contenedor de aplicaciones del J2EE soporta componentes de aplicaciones dentro de la plataforma J2EE. Un contenedor es un servicio que proporciona la infraestructura necesaria y soporte para que un componente exista y para que el componente proporcione sus propios servicios a los clientes. Un contenedor proporciona generalmente servicios a los componentes como un ambiente de ejecución compatible con Java. Los componentes de aplicación fundamentales en la plataforma de J2EE son los siguientes:

- Componentes de aplicación Java: Programas Java que se ejecutan dentro de un contenedor de aplicaciones.

- Applets: Componentes de Java que se ejecutan dentro de un contenedor de applets y que son soportados generalmente vía un navegador web.

- Servlets y JSPs: Componentes de la capa web que se ejecutan dentro de un contenedor web. Servlets y JSPs proporcionan los mecanismos para la preparación de contenido dinámico, procesando, y ajustando a un formato relacionado con la presentación.

- Componentes EJB: Componentes de negocio que se ejecutan dentro de un contenedor de EJBs (empaquetado generalmente dentro del servidor de aplicaciones). Los componentes EJB o beans empresariales, se dividen en dos tipos: beans de sesión y beans de entidad. Los beans de sesión son beans empresariales que son apropiados para el proceso o flujo de trabajo. Los beans de sesión se dividen en dos tipos: con estado y sin estado. Un bean de sesión con estado conserva el estado del cliente entre las invocaciones de métodos. Un bean de sesión sin estado no conserva el estado específico del cliente entre los métodos invocados por el cliente. Se utilizan beans de sesión sin estado cuando el estado no necesita ser almacenado entre las invocaciones de métodos y pueden ofrecer beneficios de funcionamiento sobre los beans de sesión con estado, los cuales deben utilizados cuando un cierto estado necesita ser conservado entre las

invocaciones. Las instancias de un bean de sesión pertenecen a una sesión de un solo usuario y no se comparten entre usuarios.

Se utilizan beans de entidad cuando un componente de negocios necesita ser persistido y compartido entre usuarios múltiples. La persistencia del bean de entidad se puede manejar interiormente de dos maneras: persistencia administrada por el bean (BMP) y persistencia administrada por el contenedor (CMP). Se utiliza BMP cuando el diseñador del bean implanta todos los mecanismos para persistir el estado en el bean. Se utiliza el CMP cuando el diseñador del bean no implanta los mecanismos de persistencia en el bean. En lugar de eso, el diseñador del bean traza un mapa entre los atributos del bean y el almacenamiento persistente y deja al contenedor hacer el trabajo.

La plataforma J2EE especifica los servicios estándares siguientes que cada producto de J2EE soporta. Estos servicios incluyen APIs, que cada producto de J2EE debe también proporcionar a los componentes de aplicaciones de modo que los componentes puedan tener acceso a los servicios.

- HTTP: Protocolo estándar para las comunicaciones web. Los clientes pueden tener acceso HTTP vía el paquete java.net.

- HTTP sobre capa segura de conexiones (HTTPS): Es igual que el HTTP, pero el protocolo es utilizado sobre una capa segura de conexiones.

- JDBC: Un API estándar para tener acceso a recursos de la base de datos de una forma independiente del propietario.

- JavaMail: Un API que proporciona un plataforma independiente y un protocolo independiente del marco de trabajo para construir aplicaciones de correo y mensajería en Java.

- Marco de trabajo de activación de Java (JAF): APIs para un marco de trabajo de activación que es utilizada por otros paquetes, tales como JavaMail. Los desarrolladores pueden utilizar JAF para determinar el tipo de una pieza arbitraria de datos, encapsular el acceso a él, descubrir las operaciones disponibles en él e instanciar el bean apropiado para realizar estas operaciones. Por ejemplo, JavaMail utiliza JAF para determinar qué objeto instanciar dependiendo del tipo MIME del objeto.

- Invocación de métodos remotos/Protocolo Internet Inter-ORB (RMI/IIOP): Protocolo que combina las ventajas de usar las APIs RMI y el robusto protocolo de comunicaciones de CORBA, IIOP, que se comunica con clientes CORBA que han sido desarrollados usando cualquier lenguaje compatible con CORBA.

- Lenguaje de definición de interfaz de Java (JavaIDL): Es un servicio que incorpora CORBA en la plataforma Java para proporcionar usar del estándar de interoperabilidad IDL definido por el Grupo de Administración de Objetos (OMG). Los componentes de ejecución incluyen Java ORB (Agente de Petición de Objetos) para cómputo distribuido utilizando la comunicación IIOP.

- API de transacción de Java (JTA): Es un conjunto de APIs que permiten la administración de transacciones. Las aplicaciones pueden utilizar las APIs JTA para iniciar, asignar y abortar transacciones. Las APIs JTA también permiten que el contenedor se comunique con el administrador de transacciones y permite que el administrador de transacciones se comunique con el administrador de recursos.

- JMS: Un API que se comunica con MOM para permitir la comunicación punto a punto y publicar o firmar mensajería entre sistemas. JMS ofrece independencia del propietario al utilizar MOM en aplicaciones Java.

- Interfaz Java de Nombramiento y Directorios (JNDI): Una interfaz unificada para tener acceso a diversos tipos de servicios de nombramiento y de directorios. JNDI se utiliza para registrar y mejorar los componentes de negocio y otros objetos orientados a servicios en el ambiente J2EE. JNDI incluye soporte del Protocolo Ligero de Acceso de Directorios (LDAP), el Servicio de Nombramiento de Objetos CORBA (COS) y el Registro Java RMI. [Deepak]

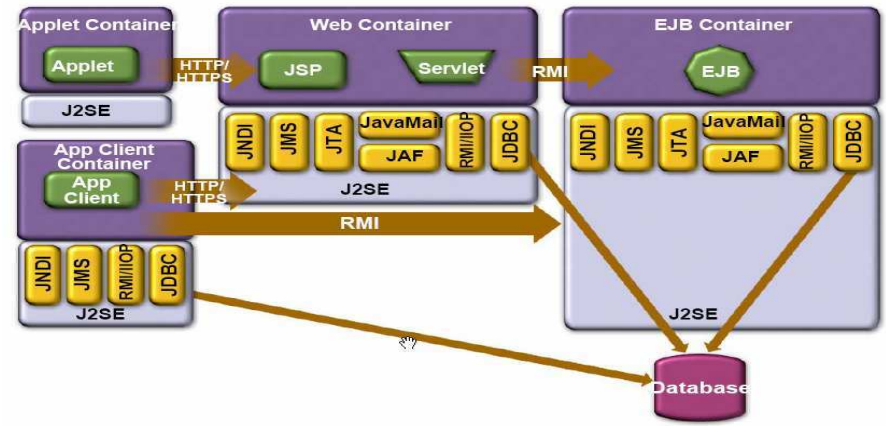



Fig. 1.5.- Diagrama de J2EE 1.4

## 1.7.- XML y Java

Antes de 1998, el intercambio de datos y de documentos estaba limitado por ser propietario o vagamente definido en formatos de documento. El advenimiento del HTML ofreció a las empresas un formato estándar para el intercambio con un enfoque en el contenido visual interactivo. Por el contrario, el HTML se define de forma rígida y no puede soportar todos los tipos de datos empresariales; por lo tanto, esos defectos proporcionaron el ímpetu para crear el XML. El estándar de XML permite a las empresas definir sus propios lenguajes de marcación con énfasis sobre tareas específicas, tales como comercio electrónico, integración de la cadena de producción, administración de datos y publicaciones.

Por estas razones, XML se ha convertido en el instrumento estratégico para definir datos corporativos a través de un número de aplicaciones de dominio. Las características de XML lo hacen conveniente para representar datos, conceptos, y contextos en plataformas abiertas y cerradas por medio de un lenguaje neutral. Utiliza etiquetas o identificadores que señalan el comienzo y el fin de un bloque de datos relacionados, para crear una jerarquía de componentes de datos relacionados llamados elementos. Alternativamente, esta jerarquía de elementos proporciona encapsulamiento y contexto. Consecuentemente, hay una mayor oportunidad de reutilizar estos datos fuera de las aplicaciones y de las fuentes de datos de las cuales fueron derivados.

Sun Microsystems, IBM, Novell, Oracle, e incluso Microsoft soportan el estándar XML. Sun Microsystems coordinó y suscribió el World Wide Web Consortium (W3C), un grupo de trabajo que desarrolló la especificación XML. Sun también creó la plataforma Java, una familia de las especificaciones que forman un desarrollo de aplicaciones y un entorno de ejecución independiente que puede trabajar en cualquier parte.

XML y las tecnologías Java tienen muchas características complementarias, y cuando son utilizados en combinación, permiten una plataforma de gran alcance para compartir y procesar datos y documentos. Aunque XML puede definir claramente datos y documentos de una manera abierta y neutral, todavía hay necesidad de desarrollar las aplicaciones que los pueden procesar. Ampliando los estándares de la plataforma Java para incluir la tecnología XML, las compañías obtendrán una solución segura a largo plazo para incluir el soporte de la tecnología XML en sus aplicaciones escritas en el lenguaje de programación Java.

El SAX (API simple para XML) es una interfaz de la tecnología Java que permite a las aplicaciones integrarse con cualquier analizador léxico de XML para recibir la notificación de los eventos del análisis. Cada vez un número mayor de analizadores léxicos basados en la tecnología Java están disponibles ahora para soportar esta interfaz.

Aquí están algunas otras formas en las que la plataforma Java apoya el estándar de XML:

- La plataforma de Java soporta intrínsecamente el estándar Unicode, simplificando el proceso de internacionalización de un documento XML. Para las plataformas sin el soporte nativo de Unicode, la aplicación debe implantar su propio manejo de caracteres de Unicode, lo que agrega complejidad a la solución total.

- La tecnología Java, que está unida al modelo del objetos de documento de W3C (DOM) provee a los desarrolladores un ambiente altamente productivo para procesar y consultar documentos XML. La plataforma Java puede convertirse en un ambiente de ejecución multiplataforma para procesar documentos de XML. El soporte intrínseco de la plataforma Java de la programación orientada a objetos significa que los desarrolladores pueden construir aplicaciones creando jerarquías de objetos Java. De forma similar, la especificación de XML ofrece una representación jerárquica de datos. Debido a que la plataforma Java y el contenido XML comparten esta característica en común, ellos son extremadamente compatibles para representar estructuras entre sí.

- Las aplicaciones escritas en el lenguaje de programación Java que procesan XML se pueden reutilizar en cualquier capa en un ambiente multicapa cliente-servidor, ofreciendo un nivel adicional de reutilización para documentos XML. No se puede decir lo mismo de ambientes de escritura o de ejecutables binarios específicos para una plataforma. [Allen]

# CAPÍTULO 2

# HERRAMIENTA

# PUMADAO

**Introducción**

El presente capítulo es fundamental para esta tesis. Nos centramos en la explicación de la herramienta que se ha desarrollado. Decimos para qué sirve, sus características principales, las distintas etapas de desarrollo como análisis, diseño, ejemplos de codificación y algunas pruebas de funcionalidad.

## 2.1.- Objetivo

El objetivo de esta herramienta, denominada **PumaDAO**, es conectarse a una base de datos contenida en un Manejador de Bases de Datos (DBMS) y obtener los metadatos (nombre de esquema, nombres de tablas, campos, tipos de datos) para así poder generar una capa de persistencia para aplicaciones Java basada en los patrones de diseño **DAO** (Data Access Objet) y **VO** (Value Object).

## 2.2.- Características

PumaDAO es una aplicación Java cuya interfaz gráfica está basada en Swing. Swing es más robusto que AWT, tiene más características y es más portable.

Para la conexión y manejo de datos se utilizan los patrones DAO y VO, de los cuales se explican sus características en el Apéndice A.

Una vez que se ha realizado la conexión a la base de datos, se muestran las tablas, los esquemas, los tipos de datos JDBC de las tablas y sus correspondientes tipos de datos Java, los cuales pueden ser modificados por el usuario.

PumaDAO genera archivos XML y clases Java. Los datos que contienen los archivos XML son generados dinámicamente en Java, construyendo un árbol DOM, que se escribe dentro de un archivo. Los archivos XML contienen los datos que se introducirán dentro de plantillas XSLT fijas. Las plantillas y los archivos XML se transforman en clases Java. El manejo de XML dentro de PumaDAO se lleva a cabo mediante JAXP (Java API for XML Processing).

JAXP es una iniciativa de Sun Microsystems para uniformizar el desarrollo de aplicaciones Java con XML, es muy importante señalar que JAXP no es un parser, sino que JAXP funciona en conjunción con un parser.

Lo que se intenta lograr mediante JAXP es interoperabilidad entre los diferentes parsers que existen en el mercado, esto es, debido a que existen diversas implementaciones de parsers se suelen definir ciertas funciones propietarias por "parser", la utilización de JAXP permite aislar a la aplicación o programa de estas funciones propietarias.

PumaDAO puede conectarse a cuatro DBMS: MySQL, SQLServer2000, PostgreSQL y Oracle 9i. Esta conexión se realiza mediante drivers JDBC de tipo 3 y 4 que proporcionan los fabricantes del DBMS.

Un controlador de tipo 3 es una librería cliente de Java puro que usa un protocolo independiente de la base de datos para comunicar las peticiones de ésta a un servidor, que se encarga después de traducir dicha petición en un protocolo de base de datos específico. La librería cliente es independiente de la base de datos actual, lo que simplifica la implantación.

Un controlador de tipo 4 es una librería de Java puro que traduce las peticiones JDBC a un protocolo de base de datos específico.

Para más información sobre PumaDAO consúltese el manual de usuario del sistema que se ubica en el CD que se proporciona con la presente tesis.

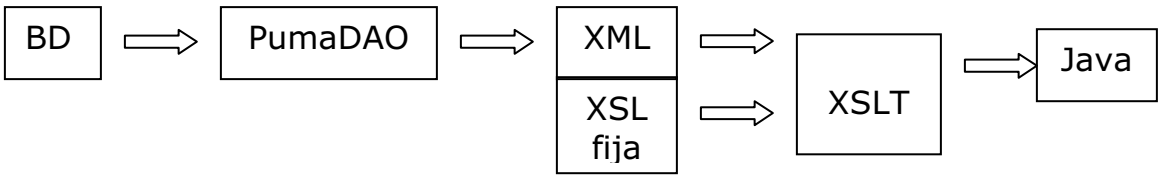



Fig. 2.1.-Diagrama de funcionamiento de PumaDAO

## 2.3.- Etapas de desarrollo

### 2.3.1.- Análisis

### Diagramas de casos de uso

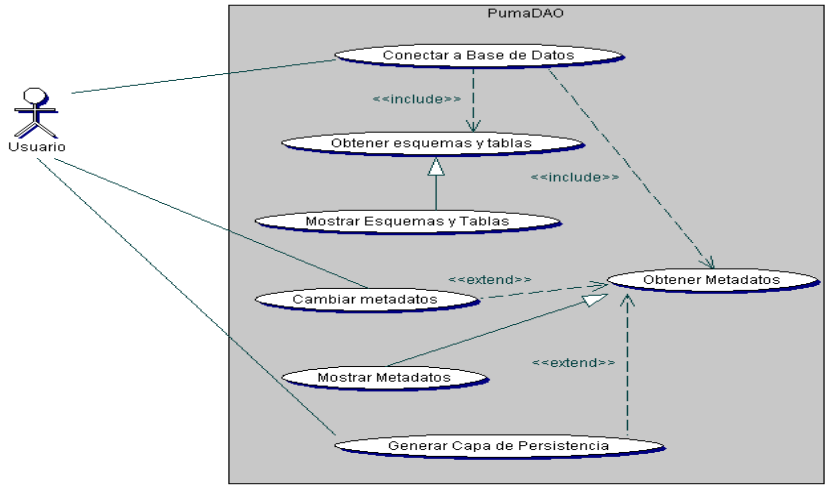



Fig. 2.2.-Caso de uso general de la herramienta PumaDAO

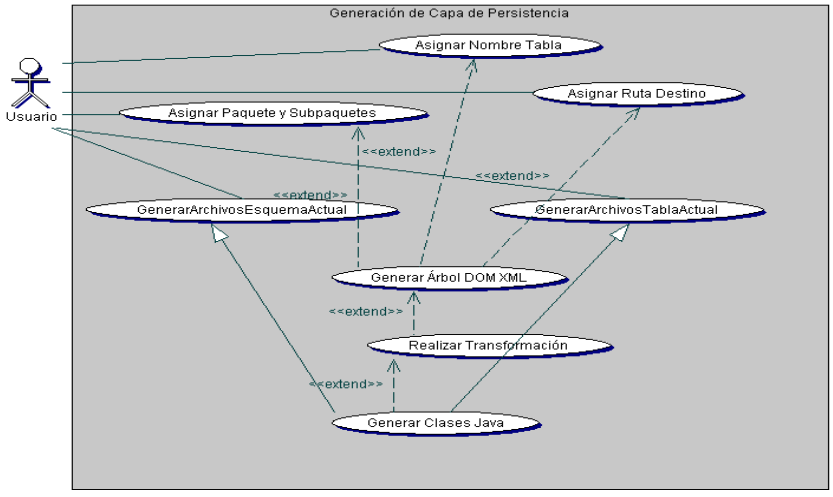



Fig.2.3.- Caso de uso de la generación de la capa de persistencia

**Diagrama de secuencias**

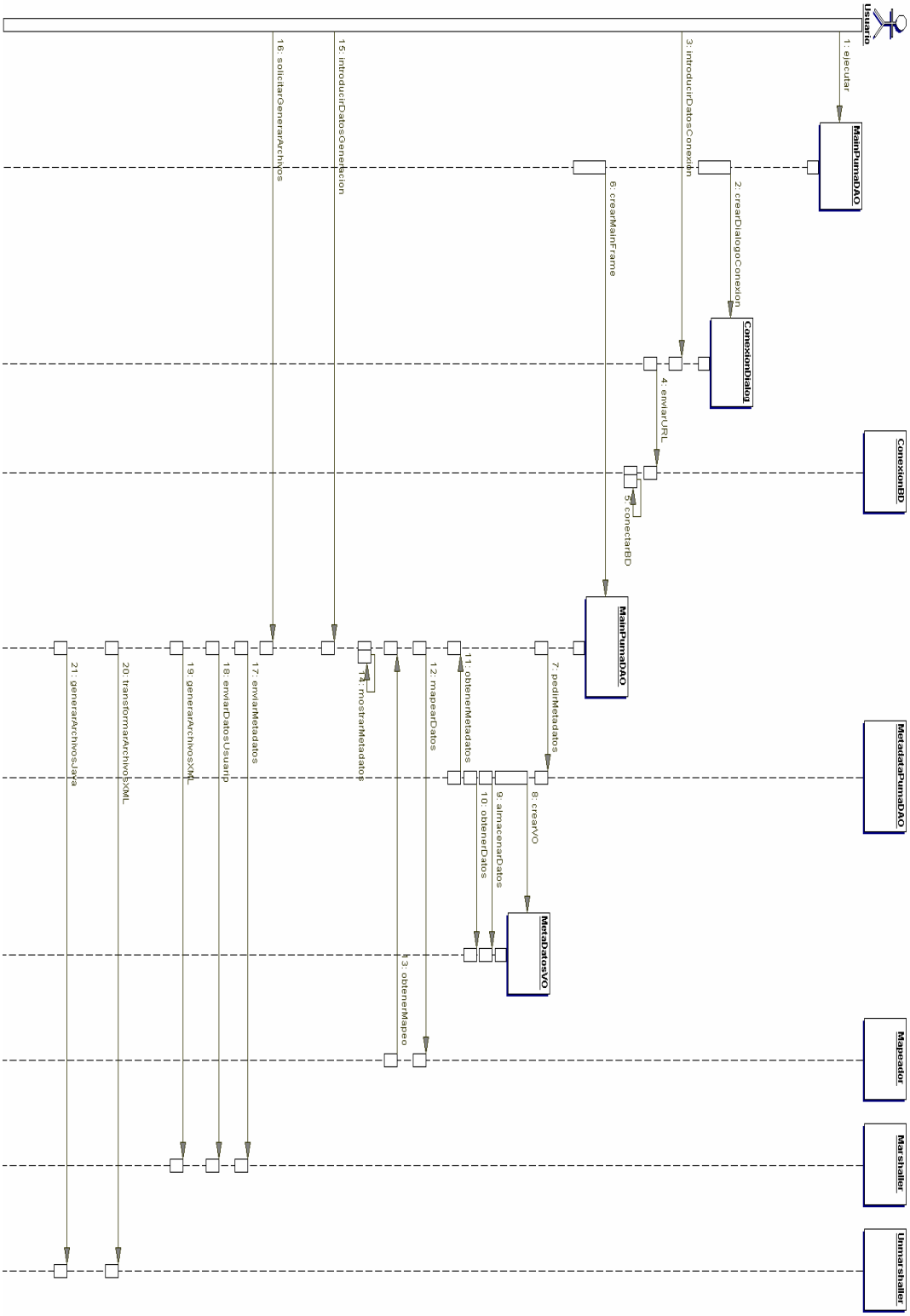



Fig. 2.4.- Diagrama de secuencias de PumaDAO

### 2.3.2.- Diseño

**Diagramas de paquetes**


Fig. 2.5.- Diagrama del paquete mx.unam.fi.pumadao

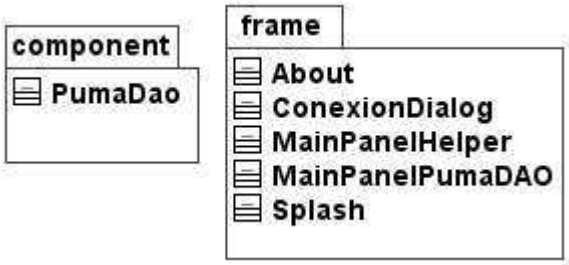



Fig. 2.6.- Diagrama del paquete mx.unam.fi.pumadao gui

**Diagramas de clases**

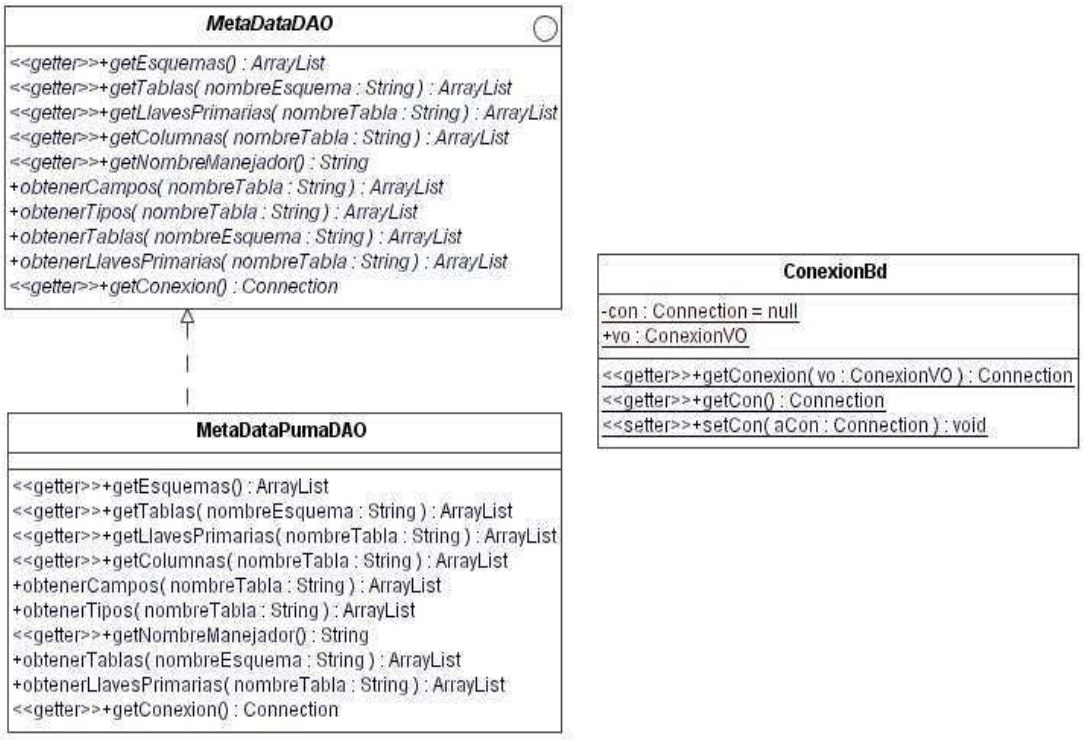



Fig. 2.7.- Diagrama de clases del paquete dao

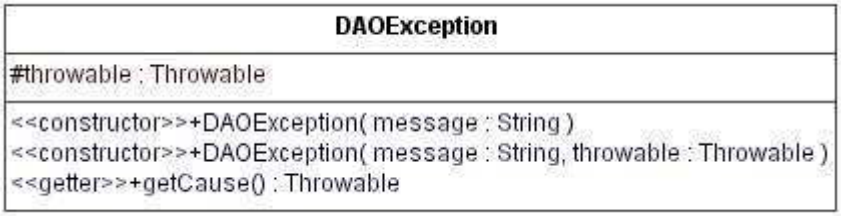



Fig. 2.8.-Diagrama de clases del paquete exception.

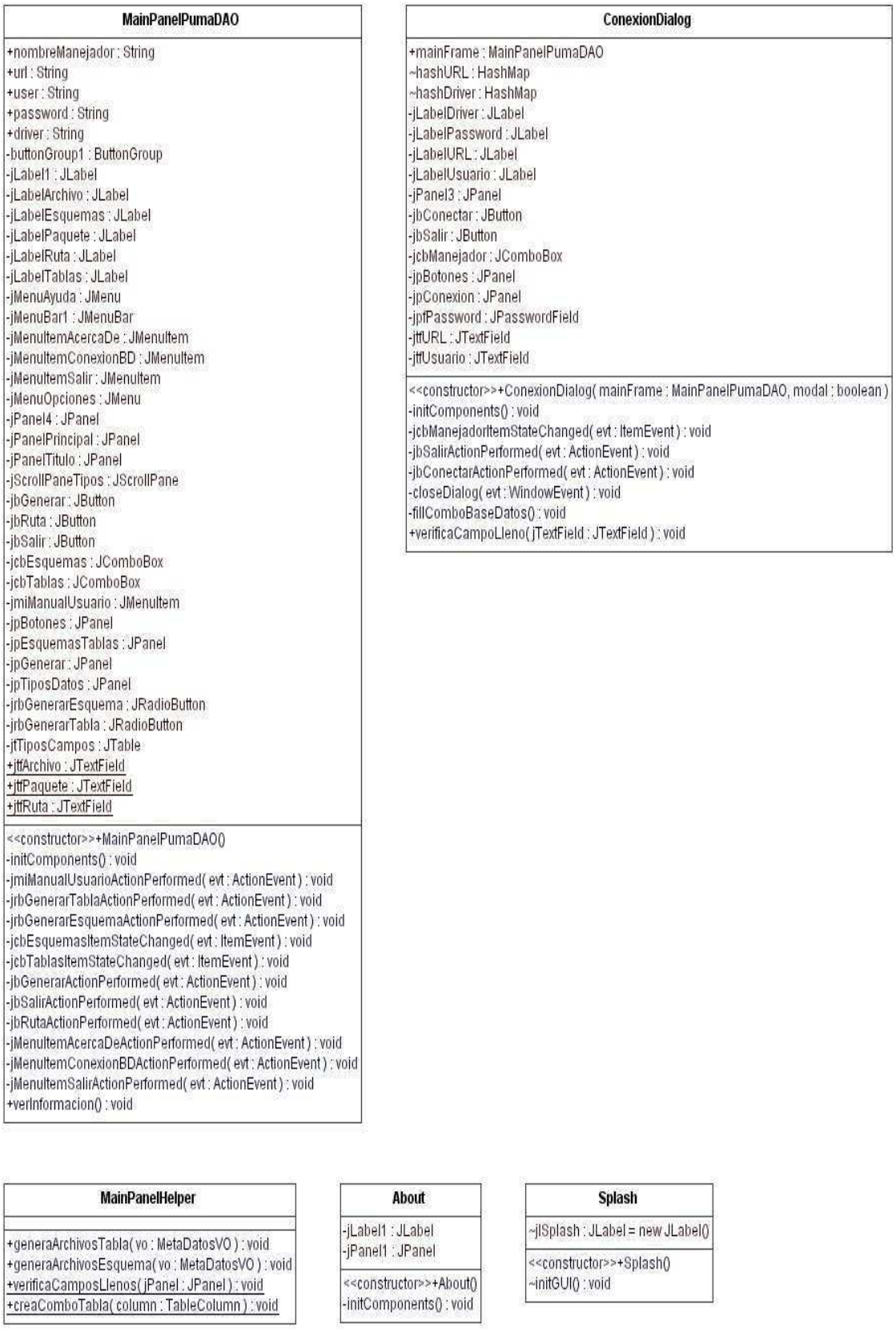



Fig. 2.9.- Diagrama de clases del paquete frame

Fig. 2.10.-Diagrama de clases del paquete marshall

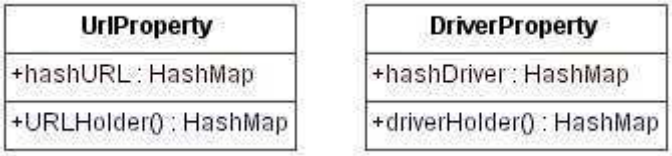



Fig. 2.11.- Diagrama de clases del paquete properties

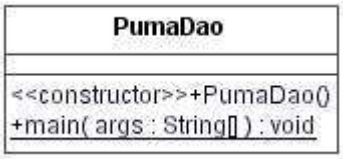



Fig. 2.12.- Diagrama de clases del paquete component

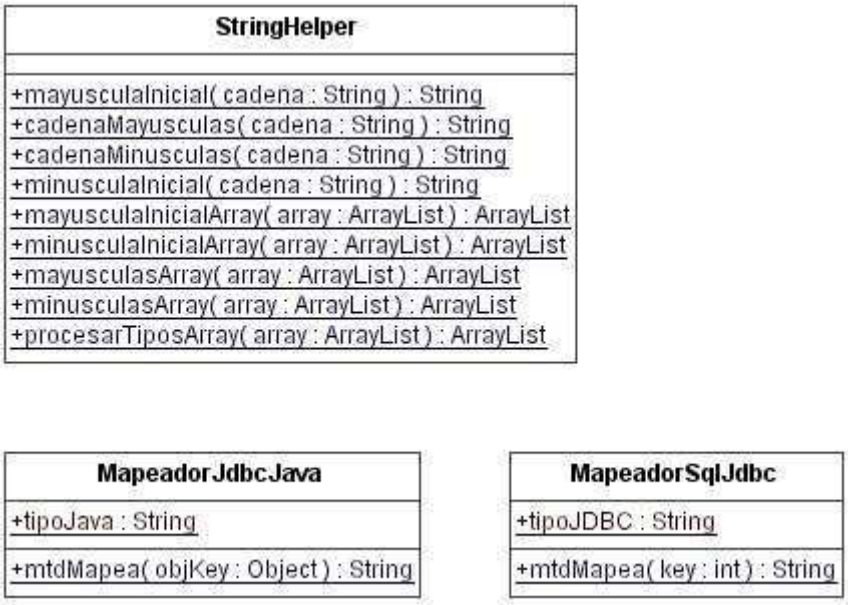



Fig. 2.13.-Diagrama de clases del paquete util

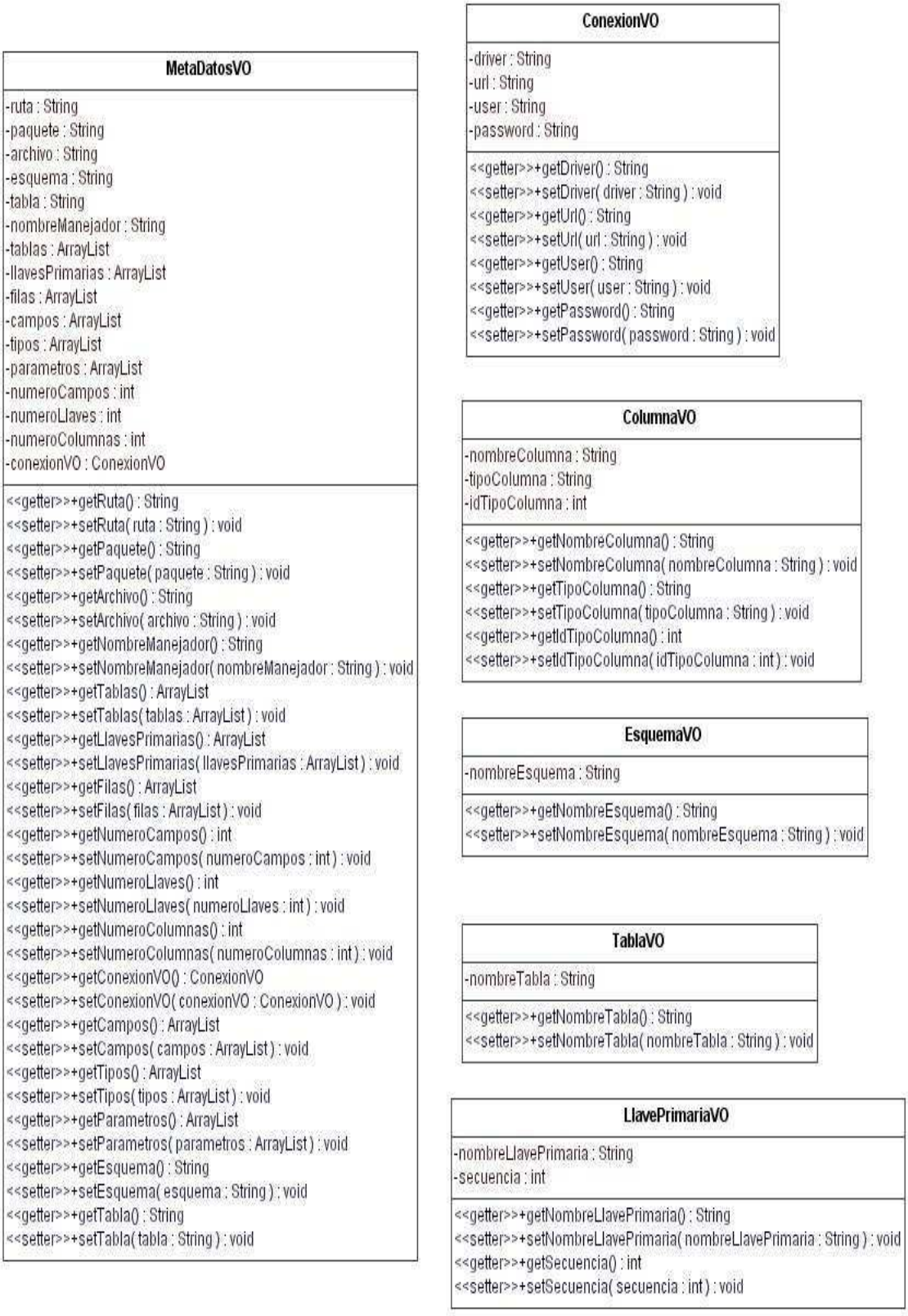



Fig. 2.14.- Diagrama de clases del paquete vo.

### 2.3.3.- Implementación

**Diagramas de componentes**

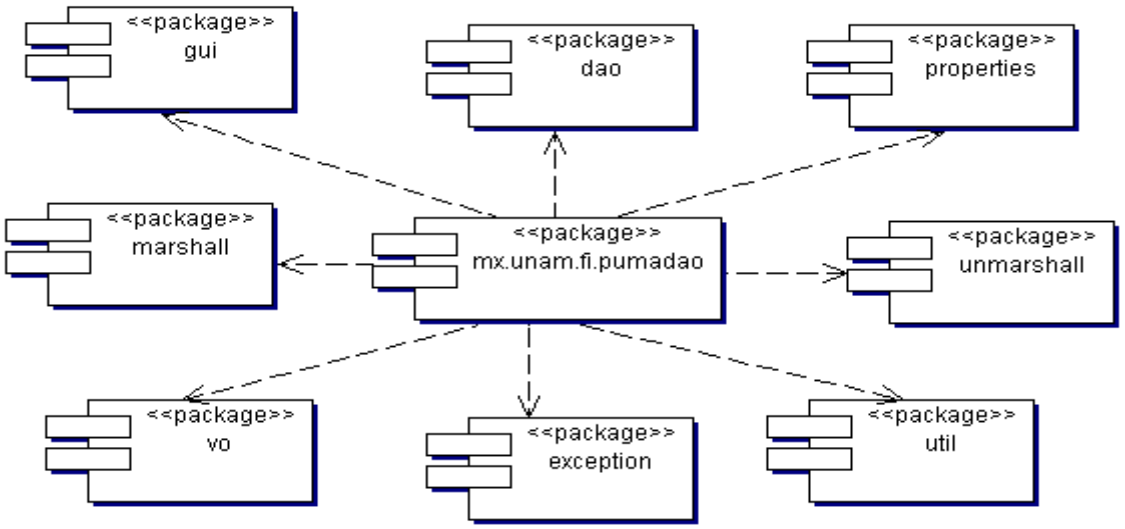



Fig. 2.15.-Organización del paquete pumadao

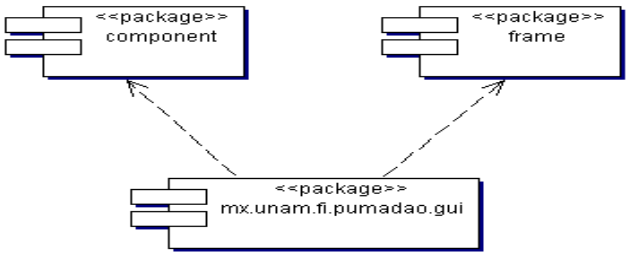



Fig. 2.16.- Organización del paquete gui

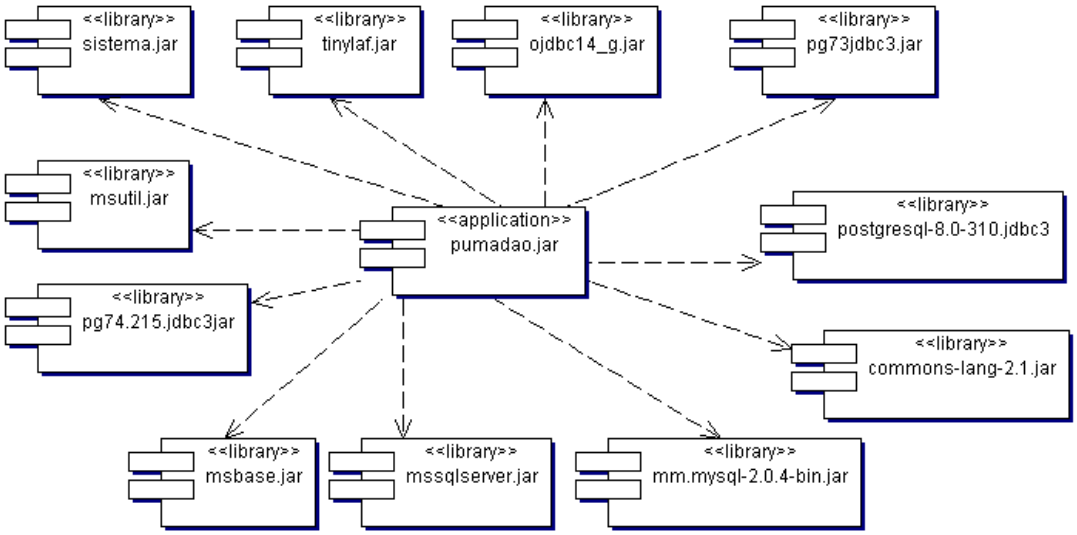



Fig. 2.17.-Componentes JAR asociados a PumaDAO

### 2.3.5.- Pruebas

Las pruebas de funcionalidad realizadas se llevaron a cabo según el siguiente diagrama:

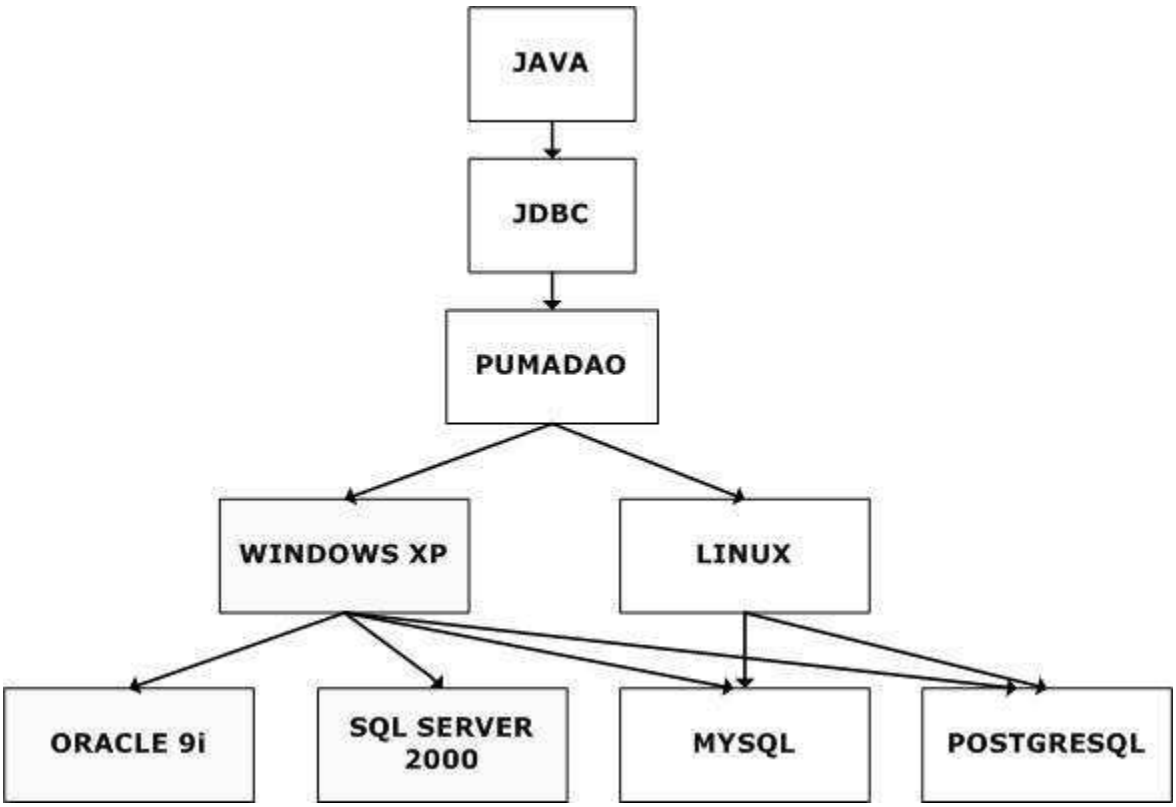



Fig. 2.18.-Diagrama de pruebas

La realización de las pruebas se hizo de la siguiente forma:

Se instalaron los manejadores de base de datos, se subió la base de datos de venta de discos que se verá en el siguiente capítulo y en la pantalla de inicio de PumaDAO se introdujeron los siguientes datos:

- Selección del manejador (por medio de un combo).
- URL de conexión a la base de datos, que contiene, el nombre del host, el puerto y el nombre da la base de datos.
- El nombre del usuario.
- El password.

Se dio clic en el botón "Conectar" y una vez dentro de la pantalla principal se seleccionaron los esquemas (si existen) y una tabla. Se verificó que los tipos de datos que aparecen en la aplicación coincidieran con los tipos de la base de datos y que se realizara correctamente el mapeo a tipo de datos Java.

Se hicieron pruebas de cambios de tipo de datos. Se generaron archivos de una y de todas las tablas de la base en distintas rutas y con distintos paquetes.

Las pruebas se hicieron tanto en Linux como en Windows.

A continuación se muestra un ejemplo de conexión por medio de Oracle 9i sobre Windows XP:

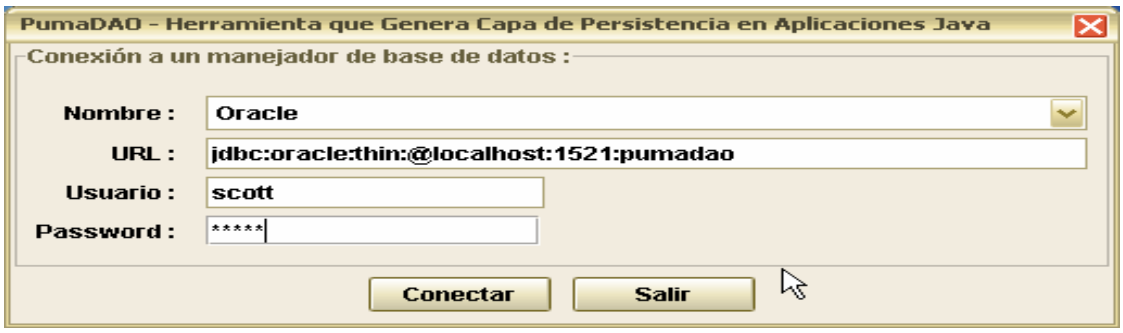



Fig. 2.19.-Diálogo de conexión a Oracle 9i en Windows XP

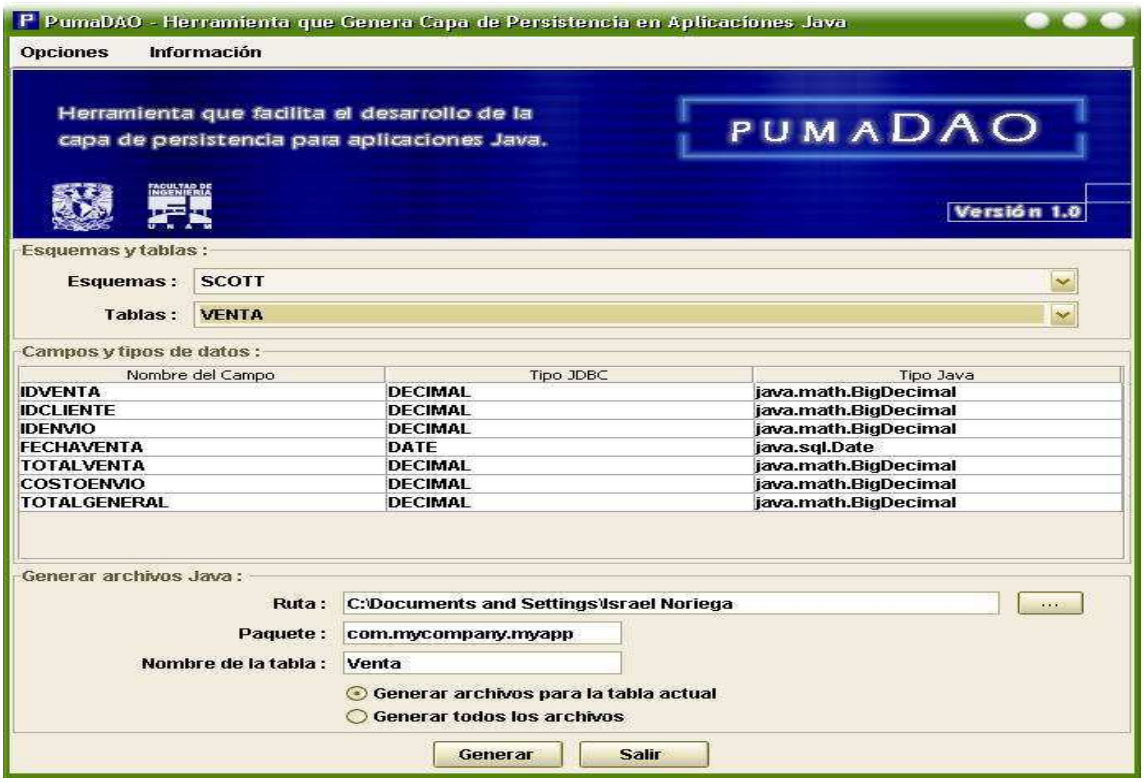



Fig. 2.20.-Conexión a Oracle 9i en Windows XP

## 2.4.- Ejemplo de archivos generados

### Interfaz DAO:

```
package com.mycompany.myapp.dao;


import com.mycompany.myapp.vo.*;
import java.sql.SQLException;

public interface VentaDao {
    public int insertVenta(Venta objVenta) throws SQLException;
    public int deleteVenta(Venta objVenta) throws SQLException;
    public Venta findVentaByPrimaryKey(int idventa) throws SQLException;
    public Venta[] findVentaByIdventa(int idventa) throws SQLException;
    public Venta[] findVentaByIdcliente(int idcliente) throws SQLException;
    public Venta[] findVentaByIdenvio(int idenvio) throws SQLException;
    public Venta[] findVentaByFechaventa(java.sql.Date fechaventa) throws
SQLException;
    public Venta[] findVentaByTotalventa(double totalventa) throws SQLException;
    public Venta[] findVentaByCostoenvio(double costoenvio) throws SQLException;
    public Venta[] findVentaByTotalgeneral(double totalgeneral) throws
SQLException;
    public Venta[] findAll() throws SQLException;
    public Venta[] findDynamicWhere(java.lang.String campo, java.lang.String
sentencia) throws SQLException;
    public int updateVenta(Venta objVenta) throws SQLException;
}
```

**Clase abstracta DAOFactory:**

```
package com.mycompany.myapp.factory;


import com.mycompany.myapp.dao.*;
import com.mycompany.myapp.factorydb.*;

public abstract class DaoFactory {

    public abstract VentaDao getVentaDao();

    public static DaoFactory getDaoFactory() {
        return new PostgresDaoFactory();
    }
}
```

**Clase concreta del manejador de base de datos:**

```
package com.mycompany.myapp.factorydb;


import java.sql.*;
import com.mycompany.myapp.dao.*;
import com.mycompany.myapp.factory.*;
import com.mycompany.myapp.sql.*;

public class PostgresDaoFactory extends DaoFactory {
    public static String DRIVER = "org.postgresql.Driver";
    public static String URL = "jdbc:postgresql://localhost:5432/discos";
    public static String USER = "administrador";
    public static String PASSWORD = "gestion";
    public static Connection con;

    public static Connection createConnection() throws Exception {
        try {
            Class.forName(DRIVER);
            con = DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (ClassNotFoundException e) {
            return null;
        }
        return con;
    }

    public VentaDao getVentaDao() {
        return new PostgresVentaDao();
    }
}
```

**Clase concreta DAO:**

```
package com.mycompany.myapp.sql;


import java.sql.*;
import java.util.*;
import com.mycompany.myapp.vo.*;
import com.mycompany.myapp.factory.*;
import com.mycompany.myapp.factorydb.*;
import com.mycompany.myapp.dao.*;

public class PostgresVentaDao implements VentaDao {
    protected static final String SQL_INSERT =
```

```
            "INSERT INTO Venta (Idventa, Idcliente, Idenvio, Fechaventa,
Totalventa, Costoenvio, Totalgeneral ) " +
            "VALUES(?, ?, ?, ?, ?, ?, ?)";

    ........
    ........


    public PostgresDaoFactory factory = new PostgresDaoFactory();

    public int insertVenta(Venta objVenta) throws SQLException {
        int count = 0;
        Connection con= null;
        PreparedStatement ps = null;

        try {
            con = factory.createConnection();
            ps = con.prepareStatement(SQL_INSERT);
            ps.setInt(1, objVenta.getIdventa());
            ps.setInt(2, objVenta.getIdcliente());
            ps.setInt(3, objVenta.getIdenvio());
            ps.setDate(4, objVenta.getFechaventa())
            ps.setDouble(5, objVenta.getTotalventa());
            ps.setDouble(6, objVenta.getCostoenvio());
            ps.setDouble(7, objVenta.getTotalgeneral());
            count = ps.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
            throw new SQLException("SQLException: " + e.getMessage());
        } catch (Exception e) {
            e.printStackTrace();
            throw new SQLException("Exception: " + e.getMessage());
        } finally {
            ConexionHelper.close(ps);
            if (con != null) {
                ConexionHelper.close(con);
            }
        }
        return count;
    }

    ........
    ........

}
```

**Clase Value Object**

```
package com.mycompany.myapp.vo;


import java.io.Serializable;

public class Venta implements Serializable {
    private int idventa;

    ........
    ........


    public int getIdventa() {
        return idventa;
    }
```

```
    public void setIdventa (int idventa) {
        this.idventa = idventa;
    }
    ........
    ........
}
```

### 2.5.- Ventajas

- Genera de una manera rápida código fuente necesario para cualquier aplicación Java. El tiempo de ahorro de codificación es grande, ya que sólo basta con oprimir un botón.
- Se pueden generar los archivos de la tabla o del esquema que se está seleccionando.
- Requiere de pocos parámetros de configuración.
- La interfaz es sencilla e intuitiva.

### 2.6.- Posibles mejoras.

- Al incluirse sentencias básicas de SQL en los archivos generados el usuario actualmente agrega de forma directa sentencias más complejas, esto puede hacerse creando nuevos objetos que se apeguen a la estructura de archivos generados. La mejora consistiría en crear de forma automática los objetos necesarios de acuerdo a las sentencias que ingrese el usuario.
- Los drivers de conexión se agregan de forma manual en las carpetas correspondientes por lo que en futuras versiones es posible implementar un módulo de carga de manejadores de bases de datos.
- Actualmente se generan métodos con las cuatro operaciones generales para un manejador de bases de datos (INSERT, DELETE, UPDATE Y SELECT) y en especial para las selecciones de datos se genera una búsqueda por cada campo que forma la tabla. La posible implementación de métodos muy específicos que solicite el usuario sería una mejora deseable.

# CAPÍTULO 3

# APLICACIÓN

# DISCNET

## 3.1.- Marco teórico

### 3.1.1.- Introducción


Figura 3.1 Menú principal del sistema DiscNet

Se ha comprobado que las aplicaciones cliente procesadas por el navegador (los applets) tienen limitaciones en tres aspectos.

- Incompatibilidad con el navegador.
- Restricciones de seguridad.
- Problemas de rendimiento debidos a largos períodos de descarga.

La aparición de la tecnología Java en el servidor ha puesto un gran cambio. Los servlets Java y las páginas Java en servidor (JSP, del inglés Java Server Pages) proporcionan una tecnología segura, sólida e independiente de la plataforma para hacer llegar la potencia de Java al comercio electrónico y al uso de la Web en la empresa.

Mediante el uso de páginas JSP, los diseñadores web y los programadores Java pueden incorporar contenido dinámico en sitios web mediante código Java embebido en las páginas JSP a través de etiquetas. Estas etiquetas proporcionan al diseñador web el modo de acceso a los datos y lógica almacenada en objetos Java, sin necesidad de ser un experto en el desarrollo de aplicaciones Java. Es más en la actualidad, proyectos como Jakarta-Taglibs proporcionan etiquetas JSP que permiten multitud de funciones, desde la conexión a una base de datos hasta la conversión al vuelo del contenido de una página para visualizarla en diferentes dispositivos físicos: computadora, teléfono móvil, PDA.

### 3.1.2.- Servlets

Desde su aparición en 1997, los servlets se han convertido en el entorno dominante de la programación Java en servidor y en un portal de uso generalizado para los servidores de aplicaciones. Los servlets aportan varias ventajas clave:

- **Rendimiento** Las tecnologías anteriores como la Interfaz de Pasarela Común CGI (Common Gateway Interface) normalmente inician un nuevo proceso para manejar cada petición que les llega. Cuando la red era simplemente un repositorio para la investigación académica y científica, no había demasiado tráfico y este sistema funcionaba bien. Los servlets, al contrario, se cargan cuando los solicitamos por primera vez y permanecen indefinidamente en la memoria. El motor de servlets carga un solo ejemplar o instancia de la clase Servlet y le lanza peticiones empleando un conjunto de subprocesos disponibles (threads o hilos). La mejora del rendimiento con este nuevo sistema es considerable.

- **Simplicidad.** Los applets Java del cliente se ejecutan en una máquina virtual proporcionada por el navegador web. Esto genera problemas de compatibilidad que incrementan la complejidad y limitan la funcionalidad de los applets. Los servlets simplifican esta situación considerablemente, ya que se ejecutan en una máquina virtual en un entorno de servidor controlado y sólo necesitan el HTTP básico para comunicarse con sus clientes. No es preciso que el cliente tenga un software especial, ni siquiera en el caso de los navegadores antiguos.

- **Sesiones HTTP.** Aunque los servidores HTTP no tienen capacidad para recordar detalles de una petición previa del mismo cliente, la interfaz API Servlet proporciona una clase HttpSession que permite superar esta limitación.

- **Acceso a la tecnología Java.** Al ser aplicaciones Java, los servlets tienen acceso directo a toda la gama de características Java, como el uso de subprocesos, acceso a redes y conectividad a bases de datos.

Las páginas JSP, que se convierten automáticamente en servlets, heredan todas estas ventajas.

#### 3.1.2.1.- Ciclo de vida de un servlet.

Como sus equivalentes applets del cliente, los servlets proporcionan métodos a los que recurrimos cuando ocurren eventos específicos en un contexto más amplio. La programación en este entorno obliga a escribir métodos predefinidos (denominados a veces métodos de retrollamada o callback), a los que se van llamando a medida que el programa que los administra lo requiere.

Los servlets operan en el contexto de un modelo de petición y respuesta administrado por el motor de servlets. El motor de servlets se encarga de:

- Cargar un servlet cuando lo solicitamos por primera vez.

- Llamar al método **init()** del servlet.

- Manejar todas las peticiones que reciba llamando al método del servlet **service().**

- Llamar al método **destroy()** de cada servlet al terminar la ejecución.

#### 3.1.2.2.- El API Servlet

Las clases e interfaces principales de la API Servlet son:

- La interfaz **Servlet.** Describe los métodos de retrollamada que se deben implementar.

- **GenericServlet.** Una clase base que implementa los métodos de la interfaz **Servlet.**

- **HttpServlet.** Una subclase específica para HTTP de GenericServlet.

- **ServletRequest.** Encapsula la información sobre la petición del cliente.

- **ServletResponse.** Proporciona acceso a un flujo de salida para que los resultados se devuelvan al cliente.

- **ServletContext**. Permite a un grupo de servlets interoperar entre sí en una aplicación Web.

### 3.1.2.3.- Visión general de las JSP

Una Página Java en Servidor es una plantilla para una página web que emplea código Java para generar un documento HTML dinámicamente. Las páginas JSP se ejecutan en un componente del servidor conocido como contenedor de JSP, que las traduce a servlets Java equivalentes.

Por esta razón los servlets y las páginas JSP están íntimamente relacionados. Lo que se puede hacer con una tecnología es, en gran medida, también posible con la otra; aunque cada una tiene capacidades propias. Como son servlets, las páginas JSP tienen todas las ventajas de los servlets:

- Tienen un mayor rendimiento y capacidad de adaptación (escalabilidad) que las secuencias de comandos CGI porque se conservan en la memoria y admiten múltiples subprocesos.
- No es necesaria una configuración especial por parte del cliente.
- Incorporan soporte para sesiones HTTP, lo que hace posible la programación de aplicaciones.
- Tienen pleno acceso a la programación Java –capacidad de reconocimiento del trabajo en red, subprocesos y conectividad a bases de datos– sin las limitaciones de los applets del cliente.

Pero además las páginas JSP tienen ventajas propias:

- Se vuelven a compilar automáticamente cuando es necesario.
- Como están en el espacio común de documentos del servidor web, dirigirse a ellas es más fácil que dirigirse a los servlets.
- Como las páginas JSP son similares al HTML, tienen mayor compatibilidad con las herramientas de desarrollo web.

### 3.1.2.4.- Cómo funcionan las JSP

Las páginas JSP pasan por tres etapas en la evolución de su código:

- **Código fuente JSP**, Este código es el que realmente escribe el desarrollador. Se encuentra en un archivo de texto con extensión .jsp y consiste en una mezcla de código de plantilla HTML, instrucciones en lenguaje Java, directivas JSP y acciones que describen como generar una página web para dar servicio a una petición concreta.
- **Código fuente Java.** El contenedor de JSP traduce el código fuente JSP al código fuente de un servlet Java equivalente. Este código fuente se guarda en un área de trabajo y suele ser útil en el proceso de depuración de errores.
- **Clase Java compilada.** Como cualquier otra clase Java, el código del servlet generado se compila en código de bytes en un archivo .class, preparado para ser cargado y ejecutado.

El contenedor de JSP administra cada una de estas etapas de la página JSP automáticamente, basándose en la situación temporal de cada archivo. Como respuesta a la petición HTTP, el contenedor comprueba si el archivo fuente .jsp ha sufrido modificaciones desde que el código fuente .java se compiló por última vez.

### 3.1.2.5.- Componentes de una página JSP

Un archivo JSP puede contener elementos JSP, datos de plantilla fijos o cualquier combinación de ambos. Los elementos JSP son instrucciones dadas al contenedor JSP sobre el código a generar y sobre cómo debe operar. Estos elementos tienen etiquetas específicas de comienzo y de fin que los identifican ante el compilador de JSP. Los datos de plantilla (normalmente HTML) son todo aquello que el contenedor de JSP no reconoce. Pasan a través del contenedor sin sufrir modificaciones, así que el HTML finalmente generado contiene los datos de plantilla tal y como estaban codificados en el archivo .jsp.

Hay tres tipos de elementos JSP:

- Directivas.
- Elementos de secuencias de comandos (scripts) que incluyen expresiones, scriplets y declaraciones.
- Acciones.

**Directivas**

Las directivas son instrucciones dirigidas al contenedor de JSP que describen el código que se debe generar. Tienen la forma genérica siguiente:

```
<%@ nombre-de-directiva [atributo=”valor” atributo=”valor” …]%>
```

La especificación JSP 1.1 describe tres directivas estándar disponibles en todos los entornos compatibles JSP: page, include y taglib.

**Directiva page (Directiva de página)**

La directiva page se emplea para especificar atributos para toda la página JSP en su conjunto. Tiene la siguiente sintaxis:

```
<%@ page [atributo=”valor” atributo=”valor” …]%>
```

**Directiva include (Directiva de inclusión)**

La directiva incluye fusiona, en el momento de la traducción, el contenido de otro archivo con el flujo de entrada del código fuente .jsp, de manera muy similar a como lo hace la directiva #include del preprocesador C. La sintaxis es:

```
<%@ incluye file=”nombre de Archivo”%>
```

**Directiva taglib**

La directiva taglib hace que las acciones personalizadas estén disponibles en la página actual mediante el uso de una biblioteca de etiquetas. La sintaxis de la directiva es:

```
<%@ taglib uri=”tagLibraryURI” prefix=”tagPrefix”%>
```

Cuyos atributos son los siguientes:

| **Atributo** | **Valor** |
|---|---|
| tagLibraryURI | La dirección URL de un descriptor de biblioteca de etiquetas (TLD, del inglés Tag Library Descriptor) |
| tagPrefix | Un prefijo único usado para identificar etiquetas personalizadas que se utilizaran |

posteriormente en la página.

**Expresiones**

JSP proporciona un medio sencillo para obtener acceso al valor de una variable Java u otra expresión y unir ese valor con el HTML de la página. La estructura sintáctica es:

```
<%= exp %>
```

**Scriplets**

Un scriplet es un conjunto de una o más sentencias o instrucciones en lenguaje Java concebidas para su uso en el procesamiento de una petición HTTP. La sintaxis de un scriplet es:

```
<% instrucción;[instrucción;…] %>
```

**Declaraciones**

Al igual que los scriplets, las declaraciones contienen instrucciones en lenguaje Java, pero con una gran diferencia: el código del scriplet se convierte en parte del método _jspService(), mientras el código de la declaración se incorpora en el archivo fuente generado fuera del método _jspService(). La sintaxis de una sección de la declaración es:

```
<%! instrucción; [instrucción; …]%>
```

Las secciones de la declaración se pueden usar para declarar clases o variables de instancia, métodos o clases internas. A diferencia de los scriplets, no tienen acceso a los objetos implícitos descritos mas adelante. Si se emplea una sección de una declaración para declarar un método que necesita utilizar el objeto de petición, por ejemplo, se debe pasar el objeto como parámetro al método.

**Acciones estándar**

Las acciones son elementos JSP de alto nivel que crean, modifican o utilizan otros objetos. A diferencia de las directivas y elementos de secuencias de comandos, las acciones están codificadas usando solamente sintaxis XML.

```
<nombre-de-etiqueta [atr=”valor” atr=”valor” …]> … </nombre-de-etiqueta>
```

### 3.1.2.6.- Arquitectura del Modelo-Vista-Controlador

La idea que subyace en el MVC es que los aspectos visuales de un sistema deberían estar aislados del funcionamiento interno, que, a su vez, debería estar separado del mecanismo que arranca y controla los elementos internos. La arquitectura MVC fue adoptada por primera vez de forma predominante por Smalltak y sus seguidores, pero ahora es un modelo de diseño muy utilizado.

El modelo se refiere al código que administra el estado interno abstracto y el funcionamiento del sistema. Administra el acceso a bases de datos y la mayor parte de la lógica de negocios. El modelo no tiene ningún componente visual, pero proporciona una interfaz de programación de aplicaciones accesible a otras partes del sistema. Esto permite escribir un programa de controlador que pueda comprobar y depurar el modelo a partir de una sencilla interfaz de línea de comandos.

La vista es la capa de presentación del sistema. No accede a base de datos y no contiene lógica de negocio. El poco código no visual que tiene la vista se limita a la lógica de presentación, por ejemplo hacer un bucle sobre un array de objetos a mostrar. Por medio del diseño, un modelo se puede asociar a más de una vista, quizás a una GUI (Interfaz Gráfica de Usuario) y a un informe por impresora. Por ejemplo en un juego basado en web podría tener una vista para cada jugador, ambas adjuntas al mismo modelo. Esto no impondrá cambios en el modelo porque este no sabe como está siendo mostrado.

El controlador es lo que manipula el modelo según la entrada del usuario. Basándose en la vista actual, en el estado del modelo y en las acciones llevadas a cabo por el usuario, el controlador invoca a la API del modelo para actualizar el estado del modelo y seleccionar la siguiente vista. En líneas generales, el controlador gestiona la entrada proveniente del usuario, mientras que la vista gestiona la salida que va hacia el usuario.

### 3.1.2.7.- Aplicaciones Web

El servidor Jakarta-Tomcat, es una aplicación web basada en Java creada para ejecutar servlets y páginas JSP, siendo la implementación oficial de referencia de las especificaciones Servlet 2.3 y JavaServer Pages 1.2.

Una aplicación web es una colección de servlets, páginas JSP, clases Java, archivos de descripción de la aplicación, documentos estáticos: HTML, XHTML, imágenes, etc. Y otros recursos que pueden ser empaquetados y ejecutados en distintos servidores de diferentes proveedores. Es decir, una aplicación web se podría definir como la capa web de cualquier aplicación.

Una de las características principales de una aplicación web es su relación con el **ServletContext.** Esta relación está controlada por el contenedor de servlets, que asocia un único **ServletContext** para cada aplicación, garantizando que las aplicaciones no van a colisionar ala hora de almacenar objetos en el **ServletContext.**

El contenedor que alberga una aplicación web no es más que la estructura de directorios en donde están colocados todos los archivos necesarios para la ejecución de la aplicación web. Por tanto, el primer paso en el desarrollo de cualquier aplicación consiste en crear la estructura de directorios en donde colocar los componentes.

En el caso de Tomcat, el directorio a partir del cual se instala cualquier aplicación web debe ser TOMCAT-HOME/webapps, en donde TOMCAT_HOME apunta al directorio de instalación de Tomcat.

Los directorios de una aplicación son :

*/aplicacionWeb*
Directorio raiz de la aplicación web, en el cual se colocan todos los archivos HTML y JSP que utiliza la aplicación. Se pueden crear subdirectorios adicionales para mantener cualquier otro recurso de tipo estático que forma parte de la aplicación web y constituyan la parte de acceso público desde cualquier navegador.

*/aplicacionWeb/WEB-INF*
Directorio que contiene todos los recursos relacionados con la aplicación web que no se han colocado en el directorio raíz y que no deban servirse al cliente. Esto es importante ya que este directorio no forma parte del documento público, por lo que ninguno de los archivos que contenga va a poder ser enviado directamente a través del servidor.

En este directorio se coloca el archivo web.xml donde se establece la configuración de la aplicación web.

*/aplicacionWeb/WEB-INF/classes*
Directorio que contiene todos los servlets y cualquier otra clase de utilidad o complementaria que se necesite para la ejecución de la aplicación web. Normalmente contiene solamente archivos .class.

*/aplicacionWeb/WEB-INF/lib*
Directorio que contiene los archivos Java de los que depende la aplicación web. Por ejemplo, si la aplicación web necesita acceso a base de datos a través de JDBC, en este directorio es en donde deben colocarse los archivos JAR que contengan el driver JDBC que proporcione el acceso a la base de datos. Normalmente solo contiene archivos .jar.

*/aplicacionWeb/WEB-INF/tlds*
Directorio que contiene los archivos TLD, descriptor de la librería de etiquetas, en el caso de que la aplicación web utilice cualquier librería de etiquetas, o acciones personalizadas.
En esta estructura, el cargador de clases consulta en primer lugar el directorio *classes* y posteriormente el directorio *lib*, de forma que en el desarrollo se pueden colocar clases de usuario en el directorio *classes* y las clases ya probadas o clases de terceros en el directorio *lib*; de este modo el cargador de clases resolverá en primer lugar las clases con las que se encuentran en desarrollo, y si no es capaz de encontrarlas en el directorio *classes* las buscara en los archivos del directorio *lib*.

Una segunda parte muy importante de toda aplicación web es su descriptor de configuración. Este descriptor no es más que un archivo XML de nombre web.xml, localizado en el directorio WEB-INF, que contiene la descripción de la configuración correspondiente a la aplicación web. En el caso anterior el archivo web.xml estará situado en el directorio /aplicacionWeb/WEB-INF. La información que contiene este descriptor puede incluir los siguientes elementos:

- Parámetros de inicialización del ServletContext
- Configuraciones de la sesión
- Definiciones de Servlets/JavaServer Pages
- Mapeado de Servlets/JavaServer Pages
- Mapeado de tipos MIME
- Configuración de seguridad
- Páginas de error
- Páginas de bienvenida

Finalmente la tercera parte importante de una aplicación web es el archivo war (Web Archive), que es el método estándar empleado para empaquetar una aplicación web y dejarla lista para su distribución y acceso a través de servidores Web con soporte para servlets y páginas JSP. Utilizando el archivo WAR se puede distribuir una aplicación web completa, compuesta por cualquier número de recursos, en una única unidad de distribución, en un único archivo.

### 3.2.- Objetivo de DiscNet

El objetivo principal de esta aplicación consiste en comprobar que las clases generadas por la herramienta PumaDAO realmente son útiles para construir cualquier desarrollo de software que involucre el acceso a datos a través de varias capas.

### 3.3.- Características de DiscNet

- Este es un sistema de ventas de discos vía internet implementado con JSP´s utilizando el patrón de diseño MVC (Modelo Vista Controlador) el cual esta formado por tres elementos: Vista (archivos JSP's), Modelo (Objetos de Negocio tales como JavaBeans representativos de cada tabla) y Controlador (Servlet Controlador que dirige las peticiones y respuestas e interactúa con las vistas y objetos de negocio).

- Puede conectarse a varias bases de datos con la misma estructura ya que la separación de las vistas (JSP´s), del controlador y de los objetos de negocio nos permite hacer uso de casi cualquier manejador de BD que pueda conectarse con Java a través del API JDBC.

- Este sistema consta de dos secciones, una donde se administran compras, clientes, ventas, pagos, envíos y otra donde un cliente puede realizar una compra de discos,

seleccionando la forma de envío y de pago. El acceso del cliente y de administrador se realiza a través de un login y password.

- El servidor de aplicaciones utilizado es Tomcat versión 4.

### 3.4.- Etapas de desarrollo de DiscNet

#### 3.4.1.- Diseño de la base de datos

El diagrama Entidad-Relación de la base de datos se muestra en la figura 3.2

#### 3.4.2.- Diagrama de flujo

El diagrama de flujo muestra en términos generales la secuencia de las vistas y acciones que realiza el sistema. En la parte de administrador únicamente indicamos los movimientos correspondientes a una tabla ya que en toda la base de datos se sigue la misma secuencia para todas las tablas. Se muestra en la figura 3.3.

#### 3.4.3.- Código de ejemplo

En esta aplicación DiscNet se utilizaron principalmente elementos JSP como directivas, scriplets y expresiones además de involucrar al código generado por la herramienta PumaDao que genera el patrón de diseño DAO.

Por ejemplo en las siguientes directivas *page* importamos en la mayoría de las páginas JSP las clases necesarias para utilizar las interfaces, clases abstractas y Value Objects que nos servirán para acceder a alguna base de datos específica.

```
<%@page import="com.discnet.dao.*"%>
<%@page import="com.discnet.factory.*"%>
<%@page import="com.discnet.vo.*"%>
```

En el scriplet siguiente se hace una búsqueda por la llave primaria en la tabla de Autores accediendo a través de las clases generadas por PumaDao.

```
<%
	DaoFactory ConcretaDF = DaoFactory.getDaoFactory();
	AutorDao objAutorDao = ConcretaDF.getAutorDao();
	String strIdAutor = request.getParameter("txtNumeroAutor");
	String strCampo = request.getParameter("rd");
	if (strIdAutor == null) {
		strIdAutor = "1";
	}
	Autor objAutor =    objAutorDao.findByPrimaryKey(strIdAutor);
%>
```

Primeramente se obtiene del método getDaoFactory() de la clase DaoFactory un objeto que representa a la fabrica concreta de objetos de la BD  por ejemplo SQLServerDaoFactory o MysqlDaoFactory dependiendo de las clases generadas por la herramienta PumaDAO y la Base de Datos seleccionada (el objeto `ConcretaDF` en este caso). Debido a que estas fabricas extienden de DaoFactory  el objeto ConcretaDF es del mismo tipo que DaoFactory.

```
DaoFactory ConcretaDF = DaoFactory.getDaoFactory();
```

Posteriormente del objeto ConcretaDF y suponiendo que es una fábrica de SQLServer por ejemplo obtenemos el método getAutorDao() (Declarado en la clase abstracta DaoFactory) y que nos devuelve un objeto del tipo SQLServerAutorDao, pero como esta clase implementa los métodos de la interfaz AutorDao son del mismo tipo.

```
AutorDao objAutorDao = ConcretaDF.getAutorDao();
```

Ahora a través del objeto objAutorDao se hace la búsqueda del objeto Autor con el método findByPrimaryKey() declarado en la interfaz e implementado en la clase SQLServerAutorDAO, que tiene además los métodos de inserción, borrado, búsqueda y modificación en la base de datos utilizando sentencias en el lenguaje SQL.

Habiendo ya hecho la búsqueda en la base de datos, podemos obtener cualquier atributo del objeto Autor a través de los métodos get() que se encuentran en su clase Value Object correspondiente y por medio de una expresión de JSP incrustar su valor dentro de algún elemento HTML correspondiente a esta página JSP, ya sea una tabla, encabezado, etc.,

```
<%=objAutor.getNombreAutor()%>
```

Vemos pues la utilidad de utilizar interfaces y clases abstractas para poder tener una transparencia en el código, pues se nota que es un código genérico y que no esta forzosamente ligado a una base de datos específica.

Con la herramienta PumaDAO se puede generar el código específico para una base de datos lo que sería un patrón de diseño Factory, pero si se genera código para cinco bases de datos específicas y se mantiene y modifica únicamente una clase DaoFactory para tener un patrón AbstractFactory, entonces crecen las posibilidades para elegir cualquier base de datos sin modificar ninguna línea en nuestras páginas JSP.

### 3.5.- Ventajas de la aplicación DiscNet

- Una de las ventajas radica en el hecho de que se pueden tener bien separadas la capa de presentación (archivos JSP's), la de negocios (clases Java generadas por PumaDAO) y la de acceso a datos (Mysql, Oracle, Postgress o SQLServer) con respecto a cualquier base que maneje el API JDBC.

- Tener implementado un Servlet Controlador que coordina el flujo de las vistas, de las peticiones y respuestas en el protocolo HTTP.

- La principal ventaja consiste en que si se genera el código de acceso a base de datos para otro manejador distinto y si se unifica una sola clase DaoFactory y un solo juego de interfaces representativas del modelo de base de datos, entonces podemos tener acceso a dos o más bases de datos.

### 3.6.- Posibles mejoras

Al ser una aplicación que sólo nos permite ejemplificar el uso de las clases generadas por PumaDAO una posible mejora consistiría en implementar elementos más complejos tales como una mayor seguridad, el uso de Enterprise Java Beans EJB´s, Web Services, etc.

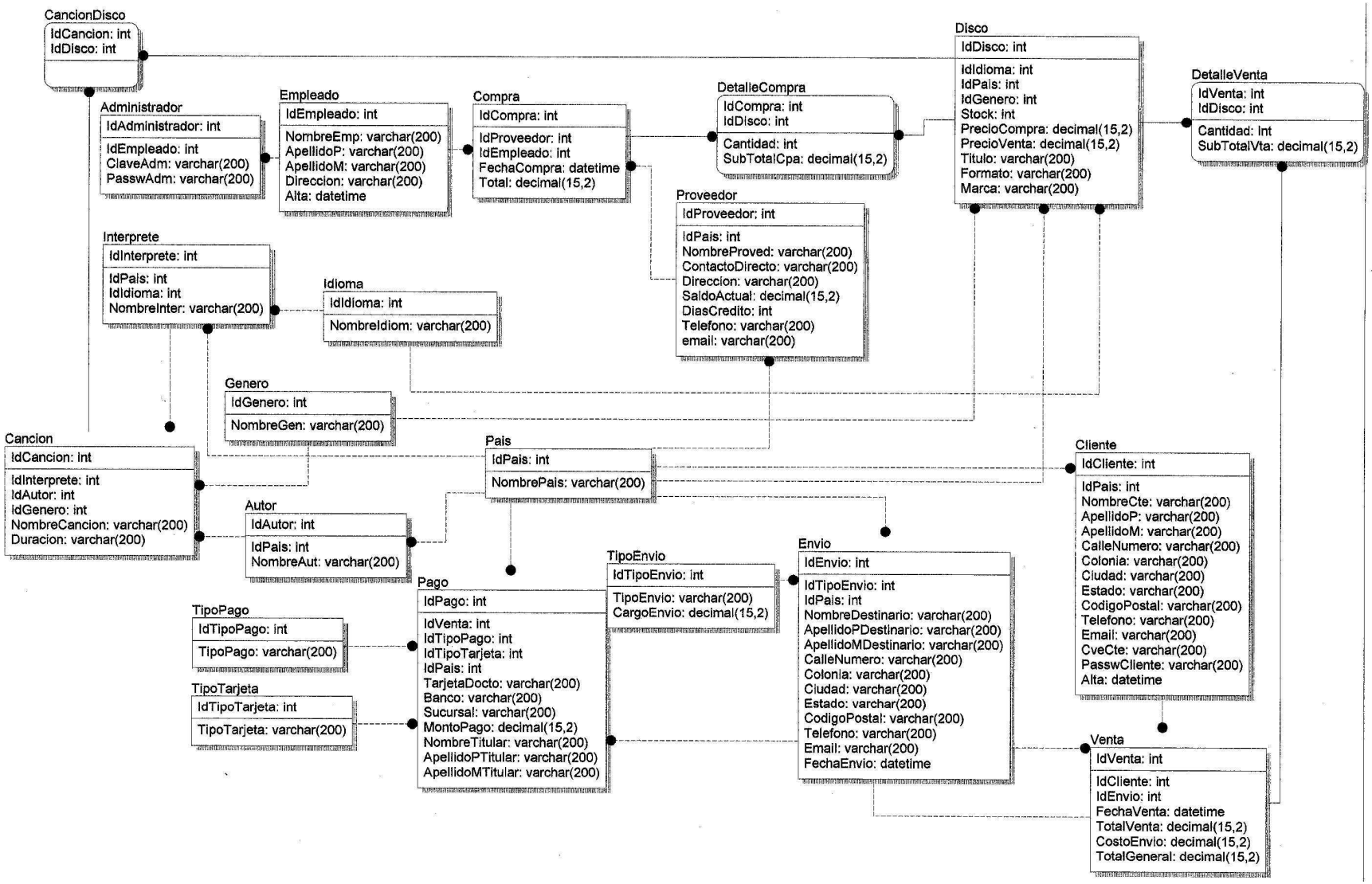



Figura 3.2. Diagrama Entidad-Relación de la BD de DiscNet

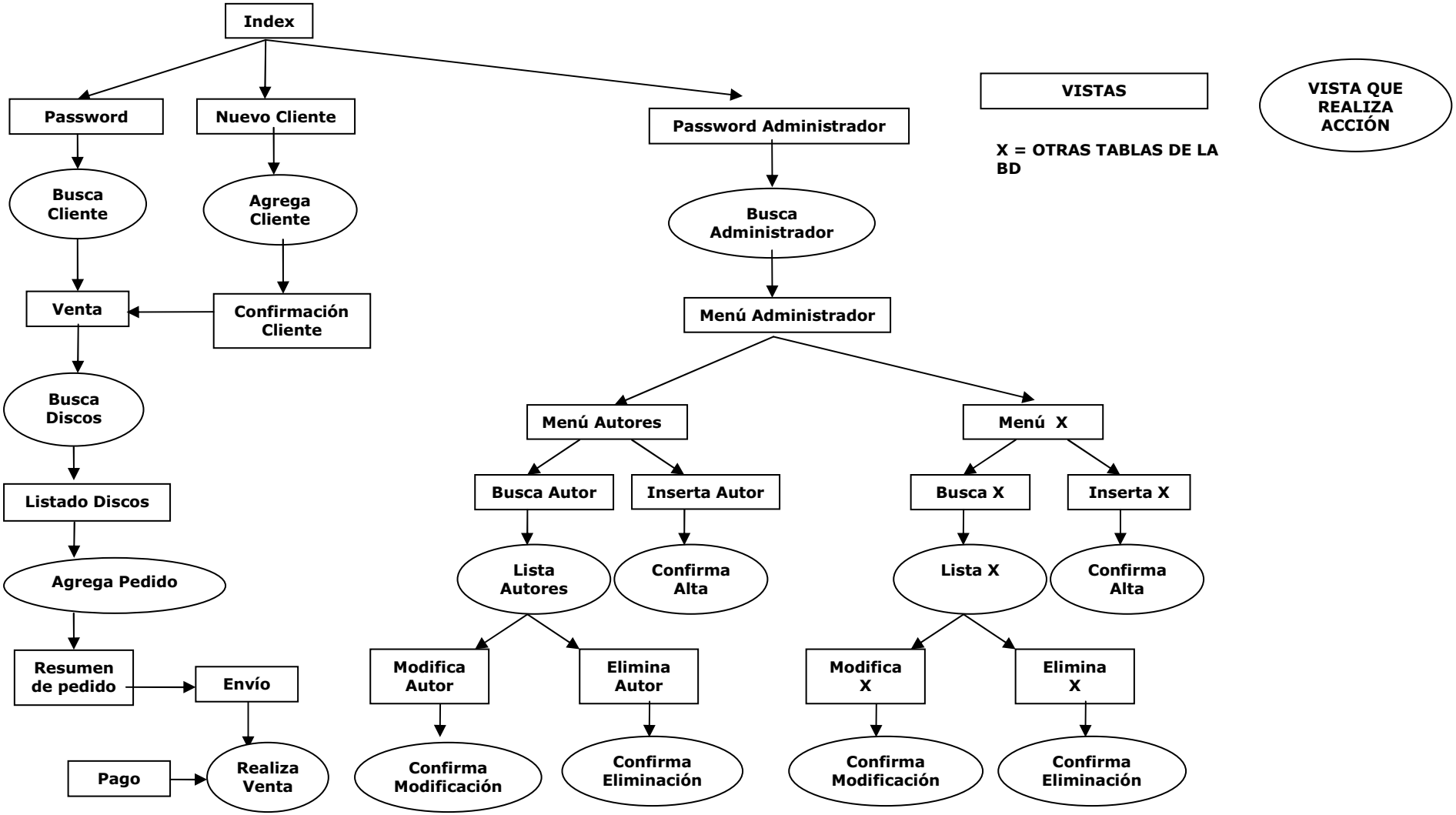



Figura 3.3. Diagrama del flujo de operaciones en DiscNet

# CONCLUSIONES

Varios conceptos fueron manejados durante la realización de este tema de tesis, entre los cuales mencionaremos los siguientes:

Aprendimos que un patrón de diseño nos permite:

- Una solución estándar para un problema común de programación.
- Una técnica para hacer flexible el código haciéndolo satisfacer ciertos criterios.
- Una estructura de implementación que logra una finalidad determinada.
- Conexiones entre componentes de programas.

Los patrones de diseño DAO y VO nos ayudaron a realizar DiscNet de manera rápida y eficiente.

Por otro lado, nuestra herramienta se puede calificar como casi imprescindible en la etapa de desarrollo de un sistema pues agiliza enormemente la generación de código para el acceso a la persistencia de datos, que en la mayoría de los casos se codifica de forma manual, siendo propenso a errores y ocasionando retrasos en los desarrollos, debido a que se tiene que estar verificando el tipo de datos de cada campo en una tabla y de forma individual. En contraste, con PumaDAO si tenemos una base de datos con varias tablas simplemente se tiene que dar click en un botón y obtener el código.

La gran flexibilidad de las API's de Java nos permitieron construir PumaDAO, principalmente JDBC, DOM y XSLT. Con JDBC obtuvimos los metadatos de la base de datos a la que nos conectemos como son los nombres de campos, tipos de datos y llaves primarias y foráneas. Con el API DOM en conjunto con Java se generaron los archivos XML representativos de cada tabla es decir se tiene un mapeo de cada tabla con su respectivo archivo XML y con el API XSLT se realizó la transformación en los archivos Java finales.

Para el Front-End de PumaDAO se utilizó Swing, que es una API muy completa para la creación de interfaces puramente Java. Se utilizaron elementos como paneles, diálogos y un splash de inicio. Se insertaron diversas imágenes creándose también una barra de menús. Se utilizaron distintos administradores de diseño como FlowLayout, GridBagLayout y BorderLayout.

Dentro del Back-End de la aplicación se utilizó el mismo patrón DAO. Se utiliza una clase dinámica para las diferentes conexiones y se utiliza un DAO que extrae los metadatos de la base de datos (esquemas, tablas, llaves primarias y nombre del manejador de base de datos) y se llenan sus respectivos VO's. Dentro del DAO de metadatos se maneja una DAOException que hereda de SQLException.

En cuanto a la utilización de metodologías en el presente trabajo la orientamos principalmente hacia Programación Extrema (XP), la cual es una metodología ágil, pues la mayoría de las prácticas mencionadas se adaptaron a nuestras necesidades de desarrollo y debido a que no es un proyecto demasiado grande (aunque no por eso menos complejo) se consideró que fue lo más adecuado.

PumaDAO es una herramienta de tipo CASE pues tiene la característica de que carga de una base de datos las características más utilizadas por los sistemas (nombres de campos, tipos de datos y llaves) y posteriormente genera un esquema de la base de datos representada en los archivos Java que se elaboran. Se clasifica dentro de las herramientas RAD de generación de código y es una ayuda importante para disminuir el tiempo de desarrollo del software pues puede ser utilizado en proyectos desarrollados en Java y con los manejadores de bases de datos Oracle 9i, SQL Server 2000, PostgreSQL y MySQL.

En nuestra experiencia laboral hemos utilizado nuestra herramienta para diferentes sistemas y comprobamos con éxito que cumple con los objetivos de facilitar el acceso a bases de datos y que además se ajusta a la separación de capas en el patrón MVC debido a que las interfaces y objetos generados realmente hacen posible que se manejen diferentes bases de datos sin

afectar a la capa de presentación. También otras personas probaron la aplicación en distintos sistemas, teniendo un resultado satisfactorio.

Una limitación de PumaDAO es que en algunas aplicaciones además de utilizar altas, bajas y cambios, también se utilizan búsquedas y consultas más complejas, por lo que el usuario debe agregarlas de forma manual en los archivos elaborados.

El software PumaDAO está sometido a la licencia GNU, la cual determina sus condiciones de uso. De forma resumida, permite el libre uso, copia y modificación del software. Todos los programas bajo esta licencia deben acompañarse del código fuente, el cual son las instrucciones legibles por el ser humano que después de ser compiladas son ejecutables por el ordenador. Con ello el programa puede ser expandido por el usuario a sus necesidades respetando, eso sí, los créditos del creador original e indicando que se trata de una modificación. Las copias y modificaciones no pueden ser restringidas, es decir, no pueden cambiar de licencia, de tal manera que el software GNU siempre lo sigue siendo.

Las materias relacionadas con esta tesis son Programación de Computadoras, Ingeniería de Programación, Estructuras de Datos, Compiladores, Bases de Datos y Temas Especiales de Computación, además se tuvo que investigar de diferentes fuentes como son Internet, libros y publicaciones en revistas.

Fueron diversos los temas generales que tuvimos que abordar, los más importantes fueron: programación orientada a objetos, lenguaje de programación Java, UML, APIs para procesar documentos XML y para acceso a datos, metodologías de programación y patrones de diseño.

Se pudo comprobar el funcionamiento de los archivos generados por PumaDAO a través de DiscNet, que es un sitio de venta de discos en línea. En DiscNet se utilizan altas, bajas, cambios y consultas a la base de datos. Los métodos utilizados en los DAO’s fueron suficientes para llevar a cabo las funcionalidades que necesitaba dicha aplicación.

Se obtuvo una primera versión de la aplicación web DiscNet y se pudieron notar algunos errores al momento de generar métodos con sentencias SQL por lo que fue de gran utilidad esta aplicación para depurar y pulir la herramienta PumaDAO.

También se cumple su objetivo inicial que es el de construir una estructura de clases para elaborar una capa de persistencia basada en los patrones de diseño DAO y VO, ya que las clases generadas son completamente útiles para el acceso y transferencia de datos en la aplicación web DiscNet.

DiscNet nos permitió aplicar también el lenguaje JavaScript, que es un lenguaje interpretado orientado a las páginas web, con una sintaxis semejante a la del lenguaje Java.

JavaScript se emplea en una página web para dotarla de funcionalidades que el HTML no puede proporcionar por sí mismo. Por lo tanto el código JavaScript debe estar integrado dentro de un documento HTML.

Asimismo, DiscNet permitió apreciar la migración de Bases de Datos ya que en una primera fase de pruebas utilizamos Mysql y posteriormente SQLServer como manejadores de bases de datos, y hubo un ajuste mínimo en algunas sentencias de consulta a la base.

La distribución de DiscNet la realizamos mediante un archivo war, que se puede instalar dentro de Tomcat y que viene dentro de la aplicación DiscNet al momento de instalarla.

Todo el código generado por la herramienta es integrado para la capa de negocio de esta aplicación de ejemplo, construyendo todos los archivos correspondientes.

# BIBLIOGRAFÍA

## BIBLIOGRAFÍA

[Allen] Allen, Paul y Bambara Joseph, Sun Certified Enterprise Architect for J2EE Study Guide, McGrawHill – Osborne, 2003 ..... 8, 15, 16, 26, 71

[Booch] Booch, Grady et. al., 'El Lenguaje de Modelado Unificado', Pearson, Madrid, 1999 ..... 6

[Ceballos] Ceballos Fco.Javier, Java 2 Curso de Programación, Alfaomega Ra-Ma, Madrid, 2000 ..... 14

[Deepak] Deepak Alur, John Crupi, Dan Malks, Core J2EE Patterns Best Practices And Design Strategies Prentice Hall, USA 2003. ..... 15, 25, 63, 65

[Deshmuk] Deshmuk, Hanumant y Malana Jignnesh, SCWD Exam Study Kit, Java Web Component Developer Ceertification, Manning, 2003 ..... 17

[DOCJDK142] Documentación del JDK 1.4.2 ..... 22

## REFERENCIAS WEB

[Horstmann] Horstmann, Cay S. y Cornell Gary, Java 2 Volumen 1: Fundamentos, Prentice Hall, 2003 ..... 19

[WebBDD] Normalización de Bases de Datos y Técnicas de diseño Barry Wise Madrid 2001 http://bulma.net/impresion.phtml?nIdNoticia=483 ..... 80

[WebCase] Herramientas Case Marcela Genero Bocco Madrid (2004) http://alarcos.inf-cr.uclm.es/doc/aplicabbdd/DASBD-HerramientasCASE.pdf ..... 10

[WebIDE] Entorno Integrado de Desarrollo Wikipedia (2006) http://es.wikipedia.org/wiki/Entorno_integrado_de_desarrollo ..... 11

[WebJava] La plataforma Java Daniel Fernández Garrido México (2003) http://www.elrincondelprogramador.com/default.asp?pag=articulos%2Fleer.asp&id=56 ..... 19

[WebMVC] J2EE Architecture Aproach 2002 Sun Microsystems, Inc. All Rights Reserved.http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/app-arch/app-arch2.html#1105855 ..... 67

[WebRAD] http://coqui.lce.org/larivera/cedu5120/Asig1/VBH.html ..... 11

[WebXML] APIs de Java para XML Juan Antonio Palos Madrid (2006) http://programacion.com/java/tutorial/apis_xml/2/ ..... 77

[WebXP] Programación Extrema Madrid 2006 http://www.geocities.com/chuidiang/metodologia/extrema.html ..... 8

# APÉNDICES

# APÉNDICE A.- Patrones de diseño.

## A.1.- Data Access Object

### Problema

Las aplicaciones pueden utilizar el API JDBC para tener acceso a los datos que residen en un sistema de base de datos relacional (RDBMS). El API JDBC permite el acceso estándar y la manipulación de datos de un almacenamiento persistente, tal como una base de datos relacional. El API JDBC permite a las aplicaciones J2EE utilizar sentencias SQL, que son los medios estándares para tener acceso a las tablas del RDBMS. Sin embargo, siempre dentro de un ambiente RDBMS, la sintaxis real y el formato de las sentencias SQL pueden variar dependiendo del producto que se esté usando.

Los mecanismos de acceso, las APIs utilizadas, y las características varían entre diversos tipos de almacenamiento persistente tales como RDBMS, bases de datos orientadas objetos, archivos planos, etc. Las aplicaciones que necesitan tener acceso a datos de un sistema local o externo (tal como una mainframe o un servicio B2B) utilizan a menudo APIs que son propietarias. Tal disparidad de fuentes de datos ofrece desafíos a la aplicación y pueden potencialmente crear una dependencia directa entre el código de la aplicación y el código de acceso de datos. Cuando los componentes de negocios (beans de entidad, beans de sesión, etc.) e incluso los componentes de presentación (servlets y objetos auxiliares para páginas JSP) necesitan tener acceso a la fuente de datos, pueden utilizar la API apropiada para conectarse y manipular la fuente de datos. Sin embargo, incluir el código de conectividad y de acceso de datos dentro de estos componentes introduce un estrecho acoplamiento entre los componentes y la implementación de la fuente de datos. Tales dependencias del código en los componentes hacen difícil y tedioso emigrar la aplicación de un tipo de fuente de datos a otra. Cuando la fuente de datos cambia, los componentes necesitan ser cambiados para manejar el nuevo tipo de fuente de datos.

### Solución

**Utilizar un objeto de acceso de datos (DAO) para sustraer y encapsular todo el acceso a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.**

El DAO implementa el mecanismo de acceso requerido para trabajar con la fuente de datos. La fuente de datos podría ser un RDBMS, un servicio externo como un intercambio B2B, un depósito como una base de datos LDAP, o un servicio de negocio alcanzado vía el protocolo CORBA, Internet Inter-ORB (IIOP) o sockets de bajo nivel. El componente de negocio que depende del DAO utiliza la interfaz más simple expuesta por el DAO a sus clientes. ***El DAO oculta totalmente los detalles de la implementación de la fuente de datos a los clientes***. Debido a que la interfaz expuesta por el DAO a los clientes no cambia cuando la implementación de la fuente de datos cambia, este patrón permite que el DAO se adapte a diversos esquemas de almacenamiento sin afectar a los clientes o a los componentes de negocio. Esencialmente, el DAO actúa como adaptador entre el componente y la fuente de datos.

### Consecuencias

- **Permitir la transparencia:** Los objetos de negocio pueden utilizar la fuente de datos sin saber los detalles específicos de la implementación de la fuente de datos. El acceso es transparente porque los detalles de la implementación se ocultan dentro del DAO.

- **Permitir una migración más fácil:** Una capa de DAOs hace más fácil que una aplicación emigre a una distinta implantación de la base de datos. Los objetos de negocio no tienen ningún conocimiento de la implantación subyacente de los datos. Así, la migración implica cambios solamente a la capa de los DAOs. Además, si se emplea la

estrategia de la fábrica (factory), es posible proporcionar una implementación concreta para cada implementación de almacenamiento.

- **Reducir complejidad en el código de los objetos del negocio:** Debido a que los DAOs manejan toda la complejidad del acceso a los datos, simplifican el código en los objetos de negocio. Todo el código relacionado con la implementación (tal como sentencias SQL) está contenido en el DAO y no en el objeto de negocio. Esto mejora la legibilidad del código y el desarrollo de la productividad.

- **Centralizar todo el acceso de los datos en una capa separada:** Debido a que todas las operaciones de acceso de datos ahora se delegan a los DAOs, la capa que separa el acceso de datos se puede ver como la capa que puede aislar el resto de la aplicación de la implantación del acceso de los datos. Esta centralización hace la aplicación más fácil de mantener y manejar.

- **No es útil para la persistencia administrada por contenedor:** Debido a que el contenedor de EJB maneja beans de entidad con persistencia administrada por contenedor (CMP), el contenedor mantiene automáticamente todo el acceso al almacenamiento persistente. Las aplicaciones que usan beans de entidad administrados por contenedor no necesitan una capa DAO, puesto que el servidor de la aplicación proporciona transparentemente esta funcionalidad. Sin embargo, los DAOs siguen siendo útiles cuando se requiere una combinación de CMP (para los beans de entidad) y BMP (para los beans de sesión y servlets).

- **Agregar una capa adicional:** Los DAOs crean una capa adicional de objetos entre el cliente y la fuente de datos que necesitamos diseñar e implantar para obtener los beneficios de este patrón. Pero para obtener estos beneficios, debemos pagarlos con esfuerzo adicional.

- **Necesidad de un diseño de jerarquía de clases:** Al usar la estrategia de la fábrica (factory), la jerarquía de las fábricas concretas y la jerarquía de los productos concretos producidos por las fábricas necesitan ser diseñados e implantados. Este esfuerzo adicional necesita ser considerado si hay suficiente justificación para tal flexibilidad. Esto aumenta la complejidad del diseño. Sin embargo, se puede elegir implementar la estrategia de la fábrica comenzando con el modelo del Método de la Fábrica primero, y después cambiar a la Fábrica Abstracta en caso de que sea necesario. [Deepak]

### A.2.- Value Object (Transfer Object)

**Problema**

Las aplicaciones de la plataforma J2EE implementan componentes de negocio del lado del servidor como beans de sesión y de entidad. Algunos métodos expuestos por los componentes de negocio devuelven datos al cliente. Algunas veces, el cliente invoca a los métodos get de un objeto de negocio varias veces para obtener todos los valores de los atributos.

Los beans de sesión representan los servicios de negocio y no se comparten entre usuarios. Un bean de sesión proporciona métodos de servicios genéricos cuando se implementan para el patrón Session Facade.

Por otro lado, los beans de entidad, son objetos transacionales, multiusuario que representan datos persistentes. Un bean de entidad expone los valores de los atributos proporcionando un método accesor (también referidos como métodos get) por cada atributo que desea exponer.

Toda llamada a método hecha al objeto de servicio de negocio, ya sea a un bean de entidad o a un bean de sesión, potencialmente es una llamada remota. Así, en una aplicación de Enterprise JavaBeans (EJB) dichas llamadas remotas usan la capa de red sin importar la proximidad del

cliente al bean, creando una sobrecarga en la red. Las llamadas a métodos del bean empresarial podría penetrar las capas de red incluso si tanto el cliente como el contenedor EJB que mantiene el bean de entidad se están ejecutando en la misma JVM (Máquina Virtual Java), el mismo sistema operativo o máquina física. Algunos vendedores podrían implementar mecanismos para reducir esta sobrecarga utilizando una aproximación de acceso más directa pasando por encima de la red.

Según se incrementa la utilización de estos métodos remotos, el rendimiento de la aplicación se puede degradar significativamente. Por lo tanto, utilizar varias llamadas a métodos get que devuelven simples valores de atributos es ineficiente para obtener valores de datos desde un bean empresarial.

**Solución**

**Utilizar un Transfer Object para encapsular los datos de negocio. Se utiliza una única llamada a un método para envíar y recuperar el Transfer Object. Cuando el cliente solicita los datos de negocio al bean empresarial, éste puede construir el Transfer Object, rellenarlo con sus valores de atributos y pasarlo por valor al cliente.**

Los clientes normalmente solicitan más de un valor a un bean empresarial. Para reducir el número de llamadas remotas y evitar la sobrecarga asociada, es mejor el Transfer Object para transportar los datos desde el bean empresarial al cliente.

Cuando un bean empresarial utiliza un Transfer Object, el cliente hace una sola llamada a un método remoto del bean empresarial para solicitar el Transfer Object en vez de numerosas llamadas remotas para obtener valores de atributos individuales. Entonces el bean empresarial construye un nuevo ejemplar Transfer Object, copia dentro los valores del objeto y lo devuelve al cliente. El cliente recibe el Transfer Object y puede entonces invocar los métodos accesores (o get) del Transfer Object para obtener los valores de atributos individuales del objeto Transfer Object. O bien, la implementación del Transfer Object podría hacer que todos los atributos fueran públicos. Como el Transfer Object se pasa por valor al cliente, todas las llamadas al ejemplar Transfer Object son llamadas locales en vez de invocaciones de métodos remotos.

**Consecuencias**

- **Simplifica el bean de entidad y la interfaz remota:** El bean de entidad proporciona un método getData() para obtener el Transfer Object que contiene los valores de los atributos. Esto podría eliminarse implementando múltiples métodos get en el bean y definiéndolos en la interfaz remota del bean. De forma similar, si el bean de entidad proporciona un método setData() para actualizar los valores de atributos del bean de entidad con una sola llamada a método, se podría eliminar implementando varios métodos set en el bean.

- **Transfiere más datos en menos llamadas remotas:** En lugar de realizar múltiples llamadas sobre la red al BusinessObject para obtener los valores de los atributos, esta solución proporciona una sola llamada a un método. Al mismo tiempo, esta única llamada obtiene una mayor cantidad de datos. Cuando consideremos la utilización de este patrón, debemos tener en cuenta el inconveniente del menor número de llamadas contra la mayor transmisión de datos por cada llamada. Alternativamente, podemos proporcionar ambos tipos de métodos: métodos get y set específicos y métodos get y set genéricos. El desarrollador puede elegir la técnica apropiada según sus requerimientos.

- **Reduce el tráfico de red:** Un Transfer Object transfiere los valores desde el bean de entidad al cliente en una llamada a un método remoto. El Transfer Object actúa como un transportista de datos y reduce el número de llamadas remotas requeridas para obtener los valores de los atributos del bean; y esto significa un mejor rendimiento de la red.

- **Reduce la duplicación de código:** Usando las estrategias Entity Inherits Transfer Object y Transfer Object Factory, es posible reducir o eliminar la duplicación de código entre el bean de entidad y el Transfer Object. Sin embargo, con el uso de la estrategia Transfer Object Factory, se podría incrementar la complejidad de la implementación. También hay un costo asociado a la pérdida de rendimiento con esta estrategia al utilizar la reflexión dinámicamente. En la mayoría de los casos, la estrategia Entity Inherits Transfer Object podría ser suficiente para cumplir nuestras necesidades.

- **Podría introducir Transfer Objects obsoletos:** Adoptando la estrategia Updatable Transfer Objects se permite al cliente realizar modificaciones en la copia local del Transfer Object. Una vez que se han completado las modificaciones, el cliente puede invocar el método setData() del bean de entidad y pasarle el Transfer Object modificado. El bean recibe las modificaciones y las mezcla con los valores que él tiene. Sin embargo, podría haber un problema con Transfer Objects obsoletos. El bean de entidad actualiza sus valores, pero no tiene en cuenta si otros clientes han solicitado el mismo Transfer Object con anterioridad. Estos clientes podrían contener en su copia local ejemplares de Transfer Object que no reflejan la copia real de los datos del bean. Como el bean no tiene en cuenta a estos clientes, es posible propagar la actualización de Transfer Objects obsoletos que mantienen otros clientes.

- **Podría incrementar la complejidad debido a la sincronización y el control de versión:** El bean de entidad mezcla los valores modificados con sus propios valores cuando los recibe en un Transfer Object de un cliente. Sin embargo, el bean debe manejar la situación donde dos o más clientes simultáneamente solicitan actualizaciones conflictivas de los valores del bean. Permitir estas actualizaciones podría resultar en conflictos de datos. El control de versión es una forma de evitar estos conflictos. El bean de entidad puede incluir como uno de sus atributos un número de versión o una fecha de última modificación. Este número de versión se copiaría en el Transfer Object. Una transacción de actualización puede resolver conflictos utilizando el atributo del número de versión. Si un cliente que contiene un Transfer Object obsoleto intenta actualizar el bean, éste puede detectar el número de versión obsoleto e informar al cliente de esta condición de error. Entonces el cliente tiene que obtener el último Transfer Object y reintentar la actualización. En casos extremos esto podría resultar en un cliente hambriento (el cliente nunca puede terminar su actualización).

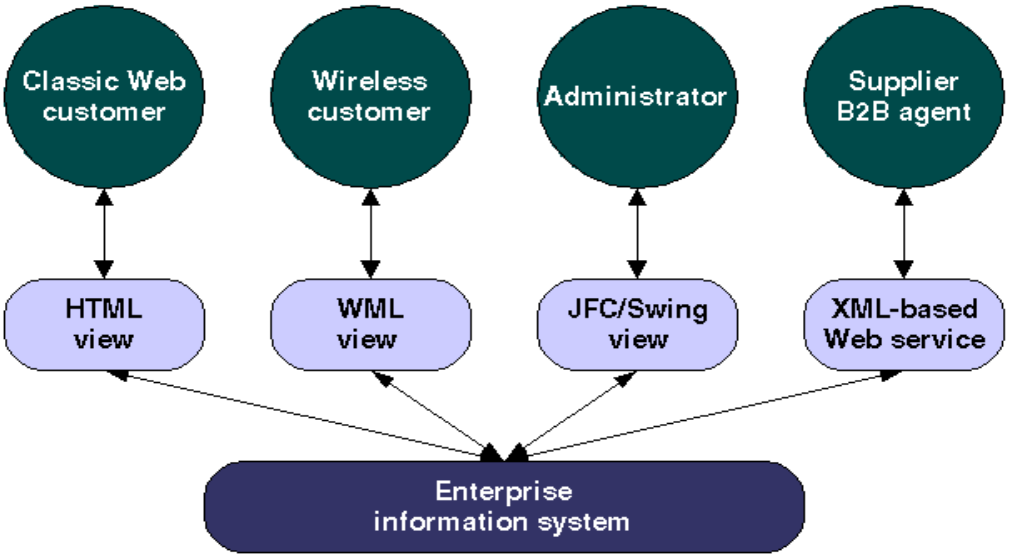
- **Accesos y transaciones concurrentes:** Cuando dos o más clientes acceden de forma concurrente al BusinessObject, el contenedor aplica la semántica de transacciones de la arquitectura EJB. Si para un bean empresarial, el nivel de aislamiento de transacciones se selecciona a TRANSACTION_SERIALIZED en el descriptor de despliegue, el contenedor proporciona la máxima protección a la transacción y se asegura de su integridad. Por ejemplo, supongamos que el flujo de trabajo para la primera transacción implica obtener un Transfer Object, luego el proceso continúa modificando el objeto BusinessObject. La segunda transacción, como se han aislado las transacciones serializadas, obtendrá el Transfer Object con los valores correctos (los más actualizados). Sin embargo, para transacciones con menos restricciones, la protección es menos rígida, permitiendo inconsistencias en el Transfer Objects por accesos concurrentes. Además, tendremos que tratar con problemas relacionados con la sincronización, el control de versión y los Transfer Objects obsoletos. [Deepak]

### A.3.- Model View Controller

#### Problema

Ahora, más que nunca, las aplicaciones empresariales necesitan soportar varios tipos de usuarios con varios tipos de interfaces. Por ejemplo, una tienda en línea podría requerir una pantalla en HTML para los clientes web, una pantalla WML para usuarios inalámbricos, una pantalla Swing para los administradores y un Web Service basado en XML para los proveedores.

Cuando desarrollamos una aplicación que soporte un solo tipo de cliente, es algunas veces benéfico entrelazar los accesos de datos y reglas de lógica de negocio con interfaces de lógica específica para presentación y control. Esto como propuesta, sin embargo, es inadecuado cuando aplicamos sistemas empresariales que necesiten soportar varios tipos de clientes. Diferentes aplicaciones necesitan ser desarrolladas para cada tipo de interfaz del cliente. El código fuera de la interfaz es duplicado en cada aplicación, resultando en esfuerzos duplicados en la implantación (frecuentemente en la forma copiar y pegar), pruebas y mantenimiento. La tarea de determinar que duplicar es costosa debido a que el código dentro y fuera de la interfaz está entrelazado. Los esfuerzos de duplicación son inevitablemente imperfectos. Lenta, pero seguramente, aplicaciones que supuestamente deben ofrecer la misma funcionalidad evolucionan hacia diferentes sistemas.




**Solución**

Aplicando el Modelo Vista Controlador (MVC) a una aplicación basada en una arquitectura Java 2 Edición Empresarial (J2EE), se puede separar la funcionalidad principal del modelo de negocios de la presentación y la lógica de control que usan esta funcionalidad. Tal separación permite múltiples vistas para compartir el mismo modelo de datos empresariales, lo cual hace que el soporte de múltiples clientes sea fácil de implantar, probar y mantener.

**Modelo (Capa de procesos de negocio)**

Modela los datos y comportamiento de los procesos de negocio

Responsable de:

- Ejecutar consultas a bases de datos
- Calcular los procesos de negocio
- Procesar órdenes

Encapsula los datos y el comportamiento que son independientes de la presentación

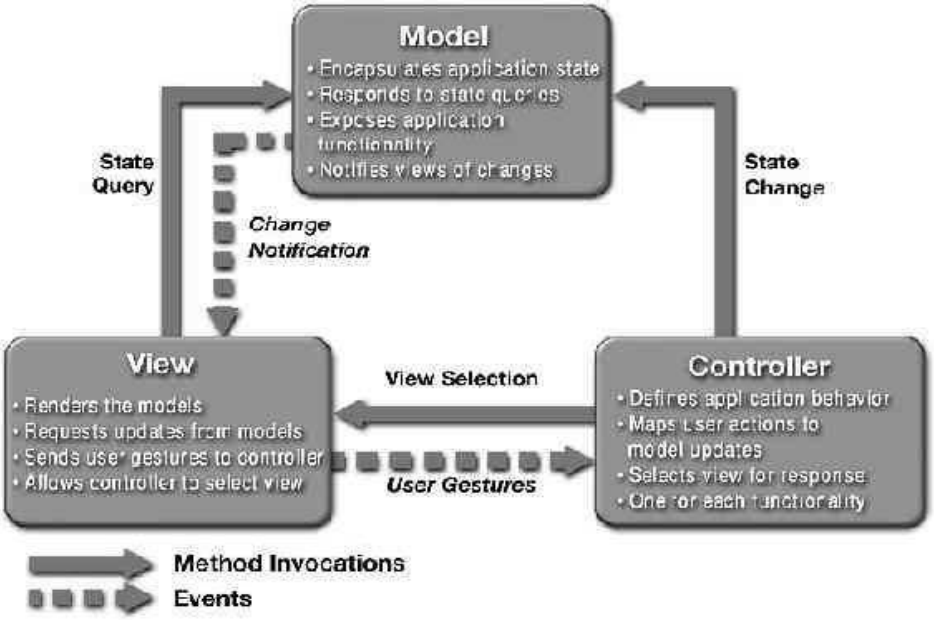
**Vista (Capa de presentación)**

- Muestra información dependiente de los tipos de clientes

- Muestra los resultados de la lógica de negocio (Modelo)
- No le importa como se obtuvo la información o de donde

**Controlador (Capa de control)**

- Sirve como conexión lógica entre la interacción del usuario y los servicios de negocio
- Responsable de tomar decisiones entre diferentes tipos de presentación
- Una petición entra a la aplicación a través de la capa de control, ella decide de que manera se manejará y que información se regresará.




**Consecuencias**

- **Reutilización de los componentes del modelo:** La separación del modelo y la vista permite que el mismo modelo empresarial utilice varias vistas. Como consecuencia, el modelo de componentes de una aplicación empresarial es más fácil de implantar, probar y mantener, debido a que todos los accesos hacia el modelo pasan a través de estos componentes.
- **Soporte más sencillo a nuevos tipos de clientes:** Para soportar un nuevo tipo de cliente, solo debe escribir una vista, realizar un poco de lógica de control y unirla a la aplicación empresarial existente.
- **Incrementa la complejidad del diseño:** Este patrón introduce algunas clases extras debido a la separación del modelo, la vista y el controlador.
- [WebMVC]

**APÉNDICE B.- Lenguaje de Modelado Unificado (UML)**

El lenguaje de modelado unificado es un lenguaje utilizado para especificación, construcción, visualización y documentación de componentes de software. El UML combina los conceptos de Booch, la Técnica de Modelado de Objetos (OMT) y la Ingeniería de Software Orientada a Objetos (OOSE). El resultado es un lenguaje estándar de modelado. Los autores de UML se enfocan en el modelado de sistemas concurrentes y distribuidos; debido a eso, UML contiene los

elementos requeridos para dirigir esos campos de acción. El UML se concentra en un modelo común que ofrece sintaxis y semántica utilizando una notación común.

**B.1.- Elementos usados en UML**

En UML un elemento es un constituyente atómico de un modelo. Un elemento de un modelo, es un elemento que representa una abstracción que está siendo dibujada del sistema que será modelado. Los elementos se usan en los diagramas UML y son los siguientes:

**Clase**

Una clase es cualquier abstracción identificada originalmente que modela una sola cosa, y el término objeto es sinónimo de instancia. Las clases tienen atributos y métodos. La clase es representada en UML por un rectángulo con tres partes. La parte del nombre es obligatoria y contiene el nombre de la clase y demás información relacionada con la documentación. Por ejemplo, el nombre podría ser *data_access_object* <<*javabean*>>. La parte del atributo es opcional y contiene las características de la clase. La parte de operaciones también es opcional y contiene definiciones de métodos. Por ejemplo: `method (argument(s)) return type: get_order (order_id) hashmap.`

**Interfaz**

Una interfaz es una colección de operaciones que representan una clase o que especifican un conjunto de métodos que deben ser implantados en la clase derivada. Una interfaz típicamente no contiene nada más que métodos virtuales y sus firmas. Java soporta interfaces directamente. La interfaz es representada en UML por un rectángulo con tres partes. La parte del nombre, la cual es obligatoria, contiene el nombre de la clase y demás información relacionada con la documentación Por ejemplo, el nombre podría ser *data_access_object* <<*javabean*>>. La parte del atributo es opcional y contiene características de la clase. La parte de operaciones también es opcional y contiene definiciones de métodos. Por ejemplo: `method (argument(s)) return type: get_order (order_id) hashmap.`

**Paquete**

Un paquete es utilizado para organizar grupos de elementos semejantes. El paquete es la única forma de agrupar elementos y su función es representar una colección de clases con funcionalidad similar. Los paquetes pueden anidarse. Los paquetes externos son algunas veces llamados dominios. Algunos paquetes externos son representados por medio de un símbolo de diapasón invertido y se les llama subsistemas. El nombre del paquete es parte del nombre de la clase. Por ejemplo, para la clase *accessdata* en el paquete *ucny.trading.com*, el nombre completo de la clase sería *ucny.trading.com.accessdata.*

**Colaboración**

La colaboración define la interacción de uno o más roles por sus contenidos, asociaciones, relaciones y clases. Para usar colaboración, los roles deben ser dirigidos hacia una clase que soporte las operaciones requeridas por el rol. El uso de la colaboración es mostrado como una elipse discontinua conteniendo el nombre de la colaboración. Una línea discontinua es dibujada desde el símbolo de la colaboración a cada uno de los objetos, dependiendo si éste aparece dentro de un diagrama de objetos que participa en la colaboración. Cada línea es etiquetada con el rol del participante.

**Caso de uso**

Un caso de uso es una descripción que representa una unidad completa de funcionalidad provista por algo tan grande como un sistema o algo tan pequeño como una clase. El resultado de esta funcionalidad es manifestado por una secuencia de mensajes intercambiados entre el sistema (o clase) y uno o más actores externos combinados con acciones realizadas por otro sistema o clase.

Hay dos tipos de casos de uso: el esencial y el real. Los casos de uso esenciales son expresados en una forma ideal que permanece libre de los detalles de la tecnología e implantación. Las decisiones del diseño son abstraídas, especialmente aquellas que se relacionan con la interfaz de usuario. Un caso de uso real describe los procesos en términos de su diseño real e implantación. Los casos de uso son esenciales al inicio del proyecto. Su propósito es ilustrar y documentar los procesos de negocios. Los casos de uso reales llegan a ser importantes después de la implantación, ya que ellos documentan la forma en como la interfaz de usuario soporta los procesos de negocio documentados en el caso de uso esencial. Un caso de uso es representado por una elipse continua conteniendo el nombre del caso de uso. Una palabra clave llamada estereotipo podría ser colocada arriba del nombre y una lista de propiedades podría ser colocada debajo del nombre.

**Componente**

El componente representa una parte modular y distribuible del sistema. Encapsula una implantación y expone un conjunto de interfaces. Las interfaces representan servicios proporcionados por elementos que residen en el componente. Un componente es típicamente distribuido en un nodo. Un componente es mostrado como un rectángulo con dos pequeños rectángulos que sobresalen en el lado izquierdo. Un componente tiene un nombre de la forma: *componente-tipo*. Una instancia de componente tiene un nombre y un tipo. El nombre del componente y su tipo pueden ser mostrados de forma subrayada dentro, arriba o abajo del símbolo del componente con la sintaxis *nombre-componente':'tipo-componente*. Ambos elementos son opcionales.

**Nodo**

El nodo es un elemento físico que representa un procesamiento de un recurso, generalmente tienen memoria y capacidad de procesamiento tal como un servidor. Obviamente, los nodos incluyen computadoras y otros dispositivos, pero también pueden ser recursos humanos o cualquier procesamiento de recursos. Los nodos pueden ser representados como tipos e instancias. Las instancias generadas en tiempo de corrida, ambos objetos e instancias de componentes pueden residir en instancias de nodo. Un nodo es típicamente representado como un cubo. Un tipo de nodo tiene un tipo de nombre: *nodo-tipo.* Una instancia de nodo tiene un nombre y un tipo de nombre. El nodo puede tener un nombre subrayado dentro o abajo del cubo. El nombre debe tener la sintaxis: *nombre':'tipo-nodo*. El nombre es el nombre del nodo individual y el tipo del nodo nos dice la clase de nodo que es.

**Estado**

El estado es una condición que puede ocurrir durante el ciclo de vida de un objeto. Puede ser también una interacción que cumpla con alguna condición, realice alguna acción o espere algún evento. Un estado compuesto tiene una descomposición gráfica. Un objeto permanece en un determinado estado por un periodo de tiempo. Un estado puede ser usado para modelar el estado de actividad interna. De forma semejante, una actividad puede ser representada como un diagrama de estados. Un estado se representa gráficamente como un rectángulo con bordes redondeados. Opcionalmente, puede tener una etiqueta con su nombre. La etiqueta del nombre es un rectángulo que contiene el nombre del estado.

### B.2.- Relaciones utilizadas en UML

| Relación | Descripción | Notación |
|---|---|---|
| Generalización (también conocida como Herencia) | Es una versión especializada de otra clase. | Una línea sólida con una punta de flecha cerrada apuntando hacia la clase más general. |
| Asociación | Utiliza los servicios de otra clase. | Una línea sólida que conecta a las clases asociadas con una punta de flecha abierta (opcional) mostrando la dirección de la |

| | | |
|---|---|---|
| | | navegación |
| Agregación | Una clase que "es dueña" de otra clase. | Una forma de asociación con un diamante vacío al final, del lado del "dueño". |
| Composición | Una clase que está compuesta de otra clase. Se refiere a una agregación dentro de la cual los componentes y la parte entera comparten tiempo de vida. | Una forma de agregación, mostrada como un diamante sólido en el extremo del "compuesto" o con el compuesto gráficamente conteniendo el "componente". |
| Refinamiento | Una versión refinada de otra clase; refinamiento dentro del cual un modelo dado puede ser mostrado como una dependencia con el estereotipo <<*refines*>> o una de sus formas más específicas como <<*implements*>> | Línea discontinua con una punta de flecha cerrada y hueca apuntando hacia la clase más refinada. |
| Dependencia | Una clase dependiente de otra clase. | Línea discontinua con una punta de flecha abierta apuntando hacia la dependencia. |

### B.3.- Diagramas utilizados en UML

**Diagrama de casos de uso**

El diagrama de caso de uso muestra actores, un conjunto de casos de uso encerrados por las fronteras del sistema, las asociaciones de comunicación o participación entre actores y los casos de uso y las generalizaciones entre los casos de uso.

**Diagrama de clases**

El diagrama de clases muestra el modelado de los elementos. También puede contener tipos, paquetes, relaciones e incluso instancias tales como objetos y enlaces. Una clase es el descriptor de un conjunto de objetos que tienen relaciones, comportamiento y estructura similar. UML proporciona la notación para declarar, especificar y utilizar las clases. Algunos elementos de modelado que son similares a las clases (tales como tipos, señales o utilerías) se conocen como estereotipos de las clases. Las clases son declaradas en el diagrama de clases y utilizadas en la mayoría de los otros diagramas.

**Diagrama de paquetes**

El diagrama de paquetes es un mecanismo utilizado para dividir y agrupar elementos del modelo, como las clases. En UML, una carpeta representa un paquete. El paquete proporciona un espacio de nombres, así que dos elementos con el mismo nombre pueden existir si los colocamos en dos paquetes separados. Los paquetes también pueden ser anidados dentro de otros paquetes. Las dependencias entre dos paquetes indican dependencias entre dos clases cualquiera entre los paquetes.

**Diagrama de estado**

El diagrama de estado es un diagrama de dos partes que muestra estados y transiciones. Muestra estados conectados por medio de transiciones. El diagrama de estado entero se une a través del modelo con una clase o un método (es decir, una operación de implantación).

**Diagrama de actividades**

Un diagrama de actividades es un caso especial de un diagrama de estados en el cual todos o la mayoría de los estados son estados de acción y en el cual todos o la mayoría de las transiciones son disparados por la finalización de una actividad en los estados origen. El diagrama de actividades se une a través del modelo a una clase o a la implantación de una operación o un caso del uso. Este diagrama se concentra en las actividades manejadas por el procesamiento interno sin tomar en cuenta las fuerzas externas. Los diagramas de actividades son utilizados en situaciones en las cuales todos o la mayoría de los eventos representan la finalización de acciones externas. Alternativamente, los diagramas de estado ordinarios se utilizan para situaciones en las cuales ocurren eventos asíncronos.

**Diagrama de secuencias**

Un diagrama de secuencia describe la forma en que los grupos de objetos colaboran en una secuencia temporal. Registra el comportamiento de un caso de uso. Muestra objetos y mensajes pasados entre los objetos de un caso de uso. Un diseño puede tener varios métodos en diferentes clases. Esto hace difícil determinar la secuencia completa del comportamiento. Este diagrama es simple y lógico, haciendo obvia la secuencia y el flujo de control.

**Diagrama de colaboración**

Un diagrama de colaboración modela las interacciones entre los objetos; los objetos interactúan invocando mensajes entre ellos. Un diagrama de colaboración agrupa las interacciones entre los diferentes objetos. Las interacciones son listadas numéricamente y ayudan a trazar la secuencia de las interacciones. Los diagramas de colaboración ayudan a identificar todas las posibles interacciones que cada objeto tiene con otros objetos.

**Diagrama de componentes**

El diagrama de componentes representan las partes de alto nivel que contiene el modelado de la aplicación. Este diagrama es la representación de alto nivel de los componentes y sus relaciones. Un diagrama de componentes representa el refinamiento de componentes después del desarrollo o en la fase de construcción.

**Diagrama de despliegue**

Un diagrama de despliegue muestra la configuración de los elementos de ejecución de la aplicación. Este diagrama obviamente es más utilizado cuando una aplicación está completa y lista para ser desplegada. [Allen]

## APÉNDICE C.- XML y XSLT

### C.1.- APIs de Java para XML

Los APIs de Java para XML nos permiten escribir aplicaciones Web completamente en el lenguaje Java. Se dividen en dos categorías: aquellas que tratan directamente con documentos XML y aquellas que tratan con procedimientos:

- Orientadas a documento:

  API Java para Procesar XML (JAXP) procesa documentos XML usando varios analizadores.

  Arquitectura Java para Uniones XML (JAXB) mapea elementos XML a clases del lenguaje Java.

- Orientadas a procedimiento:

  API Java para Mensajería XML (JAXM) envía mensajes SOAP sobre Internet de una forma estándar.

  API Java para Registros XML (JAXR) proporciona una forma estándar para acceder a registros de negocios que comparte información.

  API Java para RPC basado en XML (JAX-RPC) envía llamadas a métodos SOAP a partes remotas sobre Internet y recibe los resultados.

Quizás la característica más importante de los APIs de Java para XML es que todos soportan los estándares de la industria, así se asegura la interoperabilidad.

**El API JAXP**

El API Java para Proceso de XML (**JAXP**) hace fácil el proceso de datos XML con aplicaciones escritas en el lenguaje Java. JAXP contiene los analizadores estándar SAX (Simple API for XML Parsing) y DOM (Document Object Model) para que podamos elegir entre analizar nuestros datos como streams de eventos y construir una representación de objetos con ellos. La versión 1.1 de JAXP también soporta el estándar XSLT (XML Stylesheet Language Transformations), dándonos control sobre la representación de los datos y permitiéndonos convertir los datos a otros documentos XML o a otros formatos, como a HTML. JAXP también proporciona soporte para espacios de nombres, permitiéndonos trabajar con DTDs que de otra forma tendrían conflictos de nombrado.

Diseñado para ser flexible, JAXP nos permite usar cualquier analizador compatible XML desde dentro de nuestra aplicación. Esto lo hace con algo llamado capa de conectividad, que nos permite enchufar una implementación de los APIs SAX o DOM. La capa de conectividad también nos permite enchufar un procesador XSL, lo que nos permite controlar la forma en que se muestran los datos. La Implementación de Referencia 1.1 de JAXP proporciona el procesador de XSLT Xalan y el analizador Crimson, ambos desarrollados conjuntamente entre Sun y la Fundación de Software Apache, que proporciona software Open Source.

**El API SAX**

**SAX** define un API para un analizador basado en eventos. Estar "basado en eventos" significa que el analizador lee un documento XML desde el principio hasta el final, y cada vez que reconoce una sintaxis de construcción, se lo notifica a la aplicación que lo está ejecutando. El analizador SAX notifica a la aplicación llamando a los métodos del interfase ContentHandler. Por ejemplo, cuando el analizador encuentra un símbolo ("<"), llama al método startElement; cuando encuentra caracteres de datos, llama al método characters; y cuando encuentra un símbolo ("</"), llama al método endElement, etc. Para ilustrar, observemos el documento XML del ejemplo y veamos que hace el analizador en cada línea.

```
<priceList> [el analizador llama a startElement]
    <coffee> [el analizador llama a startElement]
        <name>Mocha Java</name> [El analizador llama a startElement, characters,
y endElement]
        <price>11.95</price> [el analizador llama a startElement, characters, y a
endElement]
    </coffee> [el analizador llama a endElement]
<priceList> [el analizador llama a endElement]
```

Las implementaciones por defecto de los métodos que llama el analizador no hacen nada, necesitamos escribir una subclase que implemente los métodos apropiados para obtener la funcionalidad que queremos. Por ejemplo, supongamos que queremos obtener el precio por libra

del café "Mocha". Escribiríamos una clase extendiendo DefaultHandler (la implementación por defecto de ContentHandler) en la que escribiríamos nuestras propias implementaciones de los métodos startElement y characters.

Primero necesitamos crear un objeto SAXParser desde un objeto SAXParserFactory. Llamaríamos al método parse sobre él, pasándole la lista de precios y un ejemplar de nuestra nueva clase handler (con sus nuevas implementaciones de los métodos startElement y characters). En este ejemplo, la lista de precios es un archivo, pero el método parse también puede aceptar una gran variedad de fuentes de entrada, incluyendo objetos InputStream, URL e InputSource.

```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser SAXParser = factory.newSAXParser();
SAXParser.parse("priceList.xml", handler);
```

El resultado de llamar al método parse depende, por supuesto, de como estén implementados los métodos en handler. El analizador SAX atravesará el archivo priceList.xml línea a línea, llamando a los métodos apropiados. Además de los métodos ya mencionados, el analizador llamará a otros métodos como startDocument, endDocument, ignorableWhiteSpace, y processingInstructions, pero estos métodos también tienen sus implementaciones por defecto que no hacen nada.

Las siguientes definiciones de métodos muestran una forma de implementar los métodos characters y startElement para que puedan encontrar e imprimir el precio del café Mocha Java. A causa de la forma en que trabaja el analizador SAX estos métodos trabajan juntos para buscar en el elemento name, los caracteres "Mocha Java", y el elemento price que sigue inmediatamente a "Mocha Java". Estos métodos usan tres banderas para seguir la pista de las condiciones que han encontrado. El analizador SAX tendrá que llamar a estos métodos más de una vez antes de se alcancen las condiciones para imprimir el precio:

```
public void startElement(..., String elementName, ...){
    if(elementName.equals("name")){
        inName = true;
    } else if(elementName.equals("price") && inMochaJava ){
        inPrice = true;
    inName = false;
    }
}

public void characters(char [] buf, int offset, int len) {
    String s = new String(buf, offset, len);
    if (inName && s.equals("Mocha Java")) {
        inMochaJava = true;
        inName = false;
    } else if (inPrice) {
        System.out.println("El precio de Mocha Java is: " + s);
        inMochaJava = false;
        inPrice = false;
    }
}
```

Una vez que el analizador ha encontrado el elemento coffee "Mocha Java", aquí tenemos el estado después de las siguientes llamadas a métodos:

- Siguiente llamada a startElement: inName es true
- Siguiente llamada a characters: inMochaJava es true
- Siguiente llamada a startElement: inPrice es true

- Siguiente llamada a characters: imprime el precio

El analizador SAX puede realizar validación mientras analiza los datos XML, lo que significa que verifica si los datos siguen las reglas especificadas en el DTD de los documentos XML. Un analizador SAX será con validación si fue creado mediante un objeto SAXParserFactory con la validación activada. Esto se hace para la fábrica de objetos SAXParserFactory en la siguiente línea de código:

```
factory.setValidating(true);
```

Para que el analizador sepa qué DTD utilizar para la validación, el documento XML debe referenciar el DTD en su declaración DOCTYPE. Esa declaración debería ser similar a esta:

```
<!DOCTYPE PriceList SYSTEM "priceList.DTD">
```

**El API DOM**

El API "Document Object Model" (**DOM**), definido por el grupo de trabajo DOM de la W3C, es un conjunto de interfaces para construir una representación de objeto, en forma de árbol, de un documento XML analizado. Una vez que hemos construido el DOM, podemos manipularlo con métodos DOM como insert y remove, igual que manipularíamos cualquier otra estructura de datos en forma de árbol. Así, al contrario que un analizador SAX, un analizador DOM permite acceso aleatorio a piezas de datos particulares de un documento XML. Otra diferencia es que con un analizador SAX, sólo podemos leer un documento XML, mientras que con un analizador DOM, podemos construir una representación objeto del documento y manipularlo en memoria, añadiendo un nuevo elemento o eliminando uno existente.

En el ejemplo anterior, usamos un analizador SAX para buscar sólo un dato en un documento. Usando un analizador DOM hubiéramos tenido que tener el modelo del objeto del documento completo en memoria, los que generalmente es menos eficiente para búsquedas que implican unos pocos elementos, especialmente si el documento es largo. En el siguiente ejemplo, añadimos un nuevo café a la lista de precios usando un analizador DOM. No podemos usar un analizador SAX para modificar la lista de precios porque sólo permite la lectura de datos.

Supongamos que queremos añadir el café "Kona" a la lista de precios. Leeremos el archivo XML de la lista de precios en un DOM y luego insertamos el nuevo elemento coffee, con su nombre y su precio. El siguiente fragmento de código crea un objeto DocumentBuilderFactory, que luego es usado para crear el objeto DocumentBuilder. Luego el código llama al método parse sobre el builder, pasándole el archivo "priceList.xml".

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse("priceList.xml");
```

En este punto, el documento es una representación DOM de la lista de precios situada en la memoria. El siguiente fragmento de código añade un nuevo café (con el nombre "Kona" y el precio de 13.50) al documento de la lista de precios. Como queremos añadir el nuevo café justo antes del café cuyo nombre es "Mocha Java", el primer paso es obtener una lista de elementos name e iterar a través de la lista para encontrar "Mocha Java". Usando el interface Node incluido en el paquete org.w3c.DOM, el código crea un objeto Node para el nuevo elemento coffee y también nuevos nodos para los elementos name y price. Estos dos elementos contienen los datos, por eso el código crea un objeto TextNode para cada uno de ellos y añade los nodos de texto a los nodos que representan los elementos name y price.

```
NodeList list = document.getElementsByTagName("name");
Node thisNode = list.getItem("name");
// loop through list
```

```
Node thisChild = thisNode.getChildNode();
if(thisNode.getFirstChild() instanceof org.w3c.DOM.TextNode) {
    String data = thisNode.getFirstChild().getData();
}
if (data.equals("Mocha Java")) {             // new node will be inserted before
Mocha Java
    Node newNode = document.createElement("coffee");
    Node nameNode = document.createElement("name");
    TextNode textNode = document.createTextNode("Kona");
    nameNode.appendChild(textNode);
    Node priceNode = document.createElement("price");
    TextNode tpNode = document.createTextNode("13.50");
    priceNode.appendChild(tpNode);
    newNode.appendChild(nameNode);
    newNode.appendChild(priceNode);
    thisNode.insertBefore(newNode, thisNode);
    }
```

Obtenemos un analizador DOM que tiene validación de la misma forma que un analizador SAX validante: llamamos a setValidating(true) sobre una fábrica de analizadores DOM antes de usarla para crear nuestro analizador DOM, y nos aseguramos de que el documento XML referencia su DTD en la declaración DOCTYPE.

**El API XSLT**

**XSLT** (XSL Transformations), definido por el grupo de trabajo XSL de la W3C, describe un lenguaje para transformar documentos XML en otros documentos XML o en otros formatos. Para realizar la transformación, normalmente necesitamos suministrar una hoja de estilo, que está escrita en "XML Stylesheet Language" (XSL). La hoja de estilo XSL específica como se mostrarán los datos XML. XSLT usa las instrucciones de formateo de la hoja de estilo para realizar la transformación. El documento convertido puede ser otro documento XML o un documento en otro formato, como HTML.

JAXP soporta XSLT con el paquete javax.xml.transform, que nos permite enchufar un transformer XSLT para realizar las transformaciones. Los subpaquetes tienen APIs de streams específicos, de SAX, y de DOM, que nos permiten realizar transformaciones directamente desde árboles DOM y eventos SAX. Los dos siguientes ejemplos muestran como crear un documento XML desde un árbol DOM y como transformar el documento XML resultante en HTML usando una hoja de estilo XSL.

**C.2.- Transformar un árbol DOM en un documento XML**

Para transformar el árbol DOM creado en la sección anterior en un documento XML, el siguiente código primero crea un objeto Transformer que realizará la transformación:

```
TransformerFactory transFactory = TransformerFactory.newInstance();
Transformer transformer = transFactory.newTransformer();
```

Usando el nodo raíz del árbol DOM, la siguiente línea de código construye un objeto DOMSource como fuente de la transformación:

```
DOMSource source = new DOMSource(document);
```

El siguiente fragmento de código crea un objeto StreamResult para tomar el resultado de la transformación y transforma el árbol en XML:

```
File newXML = new File("newXML.xml");
```

```
FileOutputStream os = new FileOutputStream(newXML);
StreamResult result = new StreamResult(os);
transformer.transform(source, result);
```

### C.3.- Transformar un documento XML en un documento HTML

También podemos usar XSLT para convertir el nuevo documento XML, "newXML.xml", a HTML usando una hoja de estilo. Cuando escribimos una hoja de estilo usamos espacios de nombres XML para referenciar el XSL construido. Por ejemplo, cada hoja de estilo tiene un elemento raíz identificando el lenguaje de la hoja de estilo, como se muestra en la siguiente línea de código:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Cuando referenciamos un constructor particular en el lenguaje de la hoja de estilos, usamos el prefijo del espacio de nombres seguido por dos puntos y el constructor particular a aplicar. Por ejemplo, la siguiente parte de una hoja de estilo indica que el dato name debe insertase en una fila de una tabla HTML:

```
<xsl:template match="name">
    <tr><td>
        <xsl:apply-templates/>
    </td></tr>
</xsl:template>
```

La siguiente hoja de estilo especifica que el dato XML es convertido a HTML y que las entradas de cafés se insertan en las filas de una tabla:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="priceList">
        <html><head>Coffee Prices</head>
            <body>
                <table>
                    <xsl:apply-templates />
                </table>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="name">
        <tr><td>
            <xsl:apply-templates />
        </td></tr>
    </xsl:template>
    <xsl:template match="price">
        <tr><td>
            <xsl:apply-templates />
        </td></tr>
    </xsl:template>
</xsl:stylesheet>
```

Para realizar la transformación necesitamos obtener un transformer XSLT y usarlo para aplicar la hoja de estilos a los datos XML. El siguiente fragmento de código obtiene un transformer instanciando un objeto TransformerFactory, lee los archivos de la hoja de estilos y del XML, crea un archivo para la salida HTML, y finalmente obtiene el objeto Transformer desde la factoría de objetos TransformerFactory llamada tFactory.

```
TransformerFactory tFactory = TransformerFactory.newInstance();
String stylesheet = "prices.xsl";
String sourceId = "newXML.xml";
File pricesHTML = new File("pricesHTML.html");
```

```
FileOutputStream os = new FileOutputStream(pricesHTML);
Transformer transformer = tFactory.newTransformer(new StreamSource(stylesheet));
```

La transformación se consigue llamando al método transform, pasándole los datos y el stream de salida:

```
transformer.transform(new StreamSource(sourceId), new StreamResult(os));
```

[REF013]

**APENDICE D.- Normalización de bases de datos y técnicas de diseño**

Básicamente, las reglas de normalización están encaminadas a eliminar redundancias e inconsistencias de dependencia en el diseño de las tablas.

Digamos que queremos crear una tabla con la información de usuarios, y los datos a guardar son el nombre, la empresa, la dirección de la empresa y algún e-mail, o bien URL si la tiene. En principio definimos la estructura de una tabla.

**D.1.- Normalización cero**

| Usuarios | | | | |
|---|---|---|---|---|
| **Nombre** | **Empresa** | **Direccion_Empresa** | **Url1** | **Url2** |
| José | ABC | Av. Urano No.55 Col. Estrella | abc.com | xyz.com |
| Martín | XYZ | Av. Cinco de Mayo No.10 Col. Independencia | abc.com | xyz.com |

Diríamos que la anterior tabla esta en nivel de normalización cero porque ninguna de nuestras reglas de normalización ha sido aplicada. Observemos los campos Url1 y Url2 ¿Qué haremos cuando en nuestra aplicación necesitemos una Url3?

Echemos un vistazo a las reglas del Primer Nivel de Normalización, y las aplicaremos a nuestra tabla.

**D.2.- Primer nivel de normalización (F/N)**

1. Eliminar los grupos repetitivos de las tablas individuales.
2. Crear una tabla separada por cada grupo de datos relacionados.
3. Identificar cada grupo de datos relacionados con una clave primaria.

La regla tres básicamente significa que tenemos que poner un campo tipo contador incremental por cada registro de la tabla original. De otra forma, ¿qué pasaría si tuviéramos dos usuarios llamados José y queremos diferenciarlos? Una vez que apliquemos el primer nivel de F/N nos encontraríamos con la siguiente tabla:

| Usuarios | | | | |
|---|---|---|---|---|
| **UserId** | **Nombre** | **Empresa** | **Direccion_Empresa** | **Url** |
| 1 | José | ABC | Av.Urano No.55 Col.Estrella | abc.com |
| 1 | José | ABC | Av.Urano No.55 Col.Estrella | xyz.com |
| 2 | Martín | XYZ | Av.Cinco de Mayo No.10 Col.Independencia | abc.com |
| 2 | Martín | XYZ | Av.Cinco de Mayo No.10 Col.Independencia | xyz.com |

Ahora diremos que nuestra tabla está en el primer nivel de F/N. Hemos solucionado el problema de la limitación del campo Url. Pero sin embargo vemos otros problemas. Cada vez que introducimos un nuevo registro en la tabla Usuarios, tenemos que duplicar el nombre de la empresa y del usuario. No sólo nuestra BD crecerá muchísimo, sino que será muy fácil que la BD se corrompa si escribimos mal alguno de los datos redundantes. Aplicaremos ahora el segundo nivel de F/N.

### D.3.- Segundo nivel de F/N

1. Crear tablas separadas para aquellos grupos de datos que se aplican a varios registros.
2. Relacionar estas tablas mediante una clave externa.

Hemos separado el campo Url en otra tabla, de forma que podemos añadir más en el futuro si tener que duplicar los demás datos. También vamos a usar nuestra clave primaria para relacionar estos campos:

| Usuarios | | | |
|---|---|---|---|
| **UserId** | **Nombre** | **Empresa** | **Direccion_Empresa** |
| 1 | José | ABC | Av. Urano No.55 Col.Estrella |
| 2 | Martín | XYZ | Av. Cinco de Mayo No.10 Col. Independencia |

| Urls | | |
|---|---|---|
| **UrlId** | **RelUserId** | **Url** |
| 1 | 1 | abc.com |
| 2 | 1 | xyz.com |
| 3 | 2 | abc.com |
| 4 | 2 | xyz.com |

Hemos creado tablas separadas y la clave primaria en la tabla Usuarios, UserId, esta relacionada ahora con la clave externa en la tabla Urls, RelUserId. Esto esta mejor. ¿Pero que ocurre cuando queremos añadir otro empleado a la empresa ABC? , ¿o 200 empleados? Ahora tenemos el nombre de la empresa y su dirección duplicándose, otra situación que puede inducirnos a introducir errores en nuestros datos. Así que tendremos que aplicar el tercer nivel de F/N.

### D.4.- Tercer nivel de F/N.

Consiste en eliminar aquellos campos que no dependan de la clave.

Nuestro nombre de empresa y su dirección no tienen nada que ver con el campo UserId, así que tienen que tener su propio EmpresaId:

| Usuarios | | |
|---|---|---|
| **UserId** | **Nombre** | **RelEmpresaId** |
| 1 | José | 1 |
| 2 | Martín | 2 |

| Empresas | | |
|---|---|---|
| **EmprId** | **Empresa** | **Direccion_Empresa** |
| 1 | ABC | Av.Urano No.55 Col. Estrella |
| 2 | XYZ | Av. Cinco de Mayo No.10 Col. Independencia |

| Urls | | |
|---|---|---|
| **UrlId** | **RelUserId** | **Url** |
| 1 | 1 | abc.com |
| 2 | 1 | xyz.com |
| 3 | 2 | abc.com |
| 4 | 2 | xyz.com |

Ahora tenemos la clave primaria EmprId en la tabla Empresas relacionada con la clave externa RelEmpresaId en la tabla Usuarios, y podemos añadir 200 usuarios mientras que sólo tenemos que insertar el nombre 'ABC' una vez. Nuestras tablas de Usuarios y Urls pueden crecer todo lo que quieran sin duplicación ni corrupción de datos. La mayoría de los desarrolladores dicen que el tercer nivel de F/N es suficiente, que nuestro esquema de datos puede manejar fácilmente los datos obtenidos de una cualquier empresa en su totalidad, y en la mayoría de los casos esto será cierto.

Pero echemos un vistazo a nuestro campo Urls ¿Vemos duplicación de datos? Esto es perfectamente aceptable si la entrada de datos de este campo es solicitada al usuario en nuestra aplicación para que teclee libremente su Url, y por lo tanto es sólo una coincidencia que José y Martín teclearon la misma Url. ¿Pero que pasa si en lugar de entrada libre de texto usáramos un menú desplegable con 20 o incluso más Urls predefinidas? Entonces tendríamos que llevar nuestro diseño de base de datos al siguiente nivel de F/N, el cuarto, muchos desarrolladores lo pasan por alto porque depende mucho de un tipo muy específico de relación, la relación 'varios-con-varios', la cual aún no hemos encontrado en nuestra aplicación.

### D.5.- Relaciones entre los datos

Antes de definir el cuarto nivel de F/N, veremos tres tipos de relaciones entre los datos: uno-a-uno, uno-con-varios y varios-con-varios. En la tabla Usuarios en el Primer Nivel de F/N del ejemplo de arriba. Por un momento imaginemos que ponemos el campo Url en una tabla separada, y cada vez que introducimos un registro en la tabla Usuarios también introducimos una sola fila en la tabla Urls. Entonces tendríamos una relación uno-a-uno: cada fila en la tabla Usuarios tendría exactamente una fila correspondiente en la tabla Urls. Para los propósitos de nuestra aplicación no sería útil la normalización.

Ahora, si se observan las tablas en el ejemplo del Segundo Nivel de F/N. Nuestras tablas permiten a un sólo usuario tener asociadas varias Urls. Esta es una relación uno-con-varios, el tipo de relación más común, y hasta que se nos presentó el dilema del Tercer Nivel de F/N. la única clase de relación que necesitamos.

La relación varios-con-varios, sin embargo, es ligeramente más compleja. En nuestro ejemplo del Tercer Nivel de F/N que tenemos a un usuario relacionado con varias Urls. Como dijimos, vamos a cambiar la estructura para permitir que varios usuarios estén relacionados con varias Urls y así tendremos una relación varios-con-varios. Veamos como quedarían nuestras tablas antes de seguir con este planteamiento:

| Usuarios | | |
|---|---|---|
| **UserId** | **Nombre** | **RelEmpresaId** |
| 1 | José | 1 |
| 2 | Martín | 2 |

| Empresas | | |
|---|---|---|
| **EmprId** | **Empresa** | **Direccion_Empresa** |
| 1 | ABC | Av.Urano No.55 Col.Estrella |
| 2 | XYZ | Av.Cinco de Mayo No.10 Col.Independencia |

| Urls | |
|---|---|
| **UrlId** | **Url** |
| 1 | abc.com |
| 2 | xyz.com |

| Url_Relations | | |
|---|---|---|
| **RelationId** | **RelatedUrlId** | **RelatedUserId** |
| 1 | 1 | 1 |
| 2 | 1 | 2 |
| 3 | 2 | 1 |
| 4 | 2 | 2 |

Para disminuir la duplicación de los datos (este proceso nos llevará al Cuarto Nivel de F/N), hemos creado una tabla que sólo tiene claves externas y primarias Url_Relations. Hemos sido capaces de remover las entradas duplicadas en la tabla Urls creando la tabla Url_Relations. Ahora podemos expresar fielmente la relación que ambos José y Martín tienen entre cada uno de ellos, y entre ambos, las Urls. Así que veamos exactamente que es lo que el Cuarto Nivel de F/N. supone:

### D.6.- Cuarto nivel de F/N.

En las relaciones varios-con-varios, entidades independientes no pueden ser almacenadas en la misma tabla.

Ya que sólo se aplica a las relaciones varios-con-varios, la mayoría de los desarrolladores pueden ignorar esta regla de forma correcta. Pero es muy útil en ciertas situaciones, tal como esta. Hemos optimizado nuestra tabla Urls eliminado duplicados y hemos puesto las relaciones en su propia tabla.

Ahora podemos seleccionar todas las Urls de José realizando la siguiente instrucción SQL:

```
SELECT NOMBRE, URL FROM USUARIOS A, URLS B, URL_RELATIONS C WHERE C.RELATEDUSERID
= 1 AND A.USERID = 1 AND B.URLID = C.RELATEDURLID
```

Y si queremos recorrer todas las Urls de cada uno de los usuarios, haríamos algo así:

```
SELECT NOMBRE, URL FROM USUARIOS A, URLS B, URL_RELATIONS C WHERE A.USERID =
C.RELATEDUSERID AND B.URLID = C.RELATEDURLID
```

### D.7.- Quinto nivel de F/N.

Existe otro nivel de normalización que se aplica a veces, pero es de hecho algo poco usual y en la mayoría de los casos no es necesario para obtener la mejor funcionalidad de nuestra estructura de datos o aplicación. Su principio sugiere:

La tabla original debe ser reconstruida desde las tablas resultantes en las cuales ha sido dividida.

Los beneficios de aplicar esta regla aseguran que no hemos creado ninguna columna extraña en las tablas y que la estructura de las tablas sea del tamaño justo que tiene que ser. Es una buena práctica aplicar esta regla, pero a no ser que estemos tratando con una extensa estructura de datos probablemente no la necesitaremos.

[WebBDD]