PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR
FACULTAD DE INGENIERÍA
ESCUELA DE SISTEMAS

ANÁLISIS Y ESTUDIO DE FRAMEWORK SALMON SOFIA COMO HERRAMIENTA DE BAJO COSTO PARA DESARROLLO DE APLICACIONES EN INTERNET UTILIZANDO PLATAFORMA JAVA J2EE Y DESARROLLO DE APLICACIÓN PROTOTIPO

TRABAJO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO DE SISTEMAS Y COMPUTACIÓN

HERNÁNDEZ HERRERA EDISON FABIÁN

QUITO, 2005

AGRADECIMIENTOS

A mis padres, que han sido mi mayor apoyo e inspiración durante toda mi vida y me han brindado la confianza para culminar mis estudios.

A todos mis amigos y a las personas que me apoyaron y ayudaron para la realización de esta tesis.

DIRECTOR DE TESIS:

Ing. Alfredo Calderón

CORRECTORES:

Ing. Rafael Melgarejo
Ing. Oswaldo Luna

TABLA DE CONTENIDO

TABLA DE CONTENIDO 4

1. INTRODUCCIÓN 7

1.1. DESARROLLO DE APLICACIONES BASADAS EN INTERNET 7

1.2. LA ARQUITECTURA WEB 9
1.2.1. Los servidores web 12
1.2.2. Las páginas web dinámicas 14

1.3. LA PLATAFORMA JAVA J2EE 16
1.3.1. Introducción al lenguaje Java 16
1.3.2. Desarrollo de aplicaciones en J2EE 19
1.3.3. Características 22
1.3.4. Arquitectura 22
1.3.5. Requerimientos 26

1.4. DESARROLLO DE APLICACIONES PARA INTERNET EN JAVA 28
1.4.1. Servlets 28
1.4.2. JavaServer Pages (JSPs) 30
1.4.3. Java Beans 31
1.4.4. Enterprise JavaBeans (EJB's) 32
1.4.5. Java Database Connectivity (JDBC) 34

2. IDE (INTEGRATED DEVELOPMENT ENVIRONMENT) 36

2.1. CONCEPTOS BÁSICOS 36

2.2. PRINCIPALES PROPUESTAS DEL MERCADO 40
2.2.1. Eclipse 40
2.2.2. IntelliJ's IDEA 48
2.2.3. Sun Java Studio 53
2.2.4. IBM WebSphere 59

2.3. VENTAJAS Y DESVENTAJAS 66

2.4. ANÁLISIS ECONÓMICO 67

2.5. COMPARACIÓN ENTRE LAS DIFERENTES PROPUESTAS 69

2.6. RESUMEN 71

3. SALMON SOFIA 73

3.1. INTRODUCCIÓN 73

3.2. DESCRIPCIÓN 75

3.3. ARQUITECTURA 79
3.3.1. MVC (Model- View-Controller) 79

3.3.2. J2EE 84
3.3.3. Framework 85

3.4. REQUERIMIENTOS 94
3.4.1. Software 94
3.4.2. Hardware 96

3.5. ANÁLISIS ECONÓMICO 97
3.5.1. Licencias 97
3.5.2. Costos 99

3.6. INSTALACIÓN Y CONFIGURACIÓN 100

3.7. CARACTERÍSTICAS 102
3.7.1. Antecedentes 102
3.7.2. Estándares y patrones 104
3.7.3. Elementos y fundamentos de SOFIA 106

3.8. DESARROLLO DE APLICACIONES EN SOFIA 112
3.8.1. Manejo de proyectos 112
3.8.2. Fundamentos de desarrollo 116
3.8.3. Componentes avanzados del SOFIA 137
3.8.4. Campos de aplicación 160

3.9. RESUMEN 161

4. DESARROLLO DE APLICACIÓN PROTOTIPO 164

4.1. CASO DE ESTUDIO: ANÁLISIS DE PRECIOS UNITARIOS 164
4.1.1. Introducción 164
4.1.2. Objetivos 166
4.1.3. Conceptos básicos en el Análisis de Precios 167
4.1.4. Necesidades y requerimientos 172

4.2. METODOLOGÍA DE DESARROLLO PARA CASO DE ESTUDIO 174
4.2.1. Análisis 175
4.2.2. Diseño 178
4.2.3. Construcción 181

5. CONCLUSIONES Y RECOMENDACIONES 184

5.1. CONCLUSIONES 184

5.2. RECOMENDACIONES 188

BIBLIOGRAFÍA 191

ANEXOS 195

ANEXO 1: MODELO ENTIDAD – RELACIÓN 195

ANEXO 2: FLUJO DE PROCESOS 196

ANEXO 3: INTERFACES DE USUARIO **200**

Capítulo 1

1. INTRODUCCIÓN

1.1. DESARROLLO DE APLICACIONES BASADAS EN INTERNET

Internet hoy en día se ha convertido en la herramienta más potente que el hombre ha inventado como medio de comunicación en el mundo. Mediante internet se puede mostrar todo tipo de información como texto, imágenes, videos, sonidos, etc. En la actualidad internet es un medio muy difundido debido a su facilidad de uso y disponibilidad.

En los últimos años, producto de la globalización y el avance de los medios de comunicación, el internet ha surgido como el 'boom' de fines del siglo XX. Su facilidad de manejo y versatilidad en pocos años ha rebasado los límites de lo inimaginable, haciéndose cada vez mayor la cantidad de usuarios que acceden a esta red y que se benefician de su enorme variedad de servicios.

Aunque en nuestro país el internet no esta totalmente difundido en comparación con otros países, gran número de empresas se están dando cuenta de las grandes posibilidades y oportunidades que ofrece el desarrollo de aplicaciones para internet.

Con la introducción de internet, se han abierto infinidad de posibilidades en cuanto al acceso a la información desde casi cualquier sitio, esto representa un desafío para los desarrolladores de aplicaciones, ya que los avances en tecnología demandan cada vez aplicaciones más rápidas, ligeras y robustas.

El lenguaje de programación CGI (Common Gateway Interface) fue el primero en añadir interactividad a las páginas de internet. Posteriormente surgieron herramientas de desarrollo más potentes como ASP (Active Server Pages), PHP (PHP Hypertext Pre-processor) o JSP (Java Server Pages). Adicionalmente han surgido otras tecnologías como el HTML dinámico (DHTML) el cual está compuesto de código HTML, hojas de estilo en cascada (Cascade Style Sheets, CSS) o scripts de programación.

En esta línea, los navegadores han ido un poco más allá, y permiten la visualización de documentos XML (eXtensible Markup Language) mediante el uso de hojas de estilo extensibles XSL (eXtensible Style Sheets). De esta manera se puede elegir que elementos visualizar, dependiendo de las circunstancias. Además existen tecnologías como JavaScript o Applets Java para agregar dinámica a una página.

1.2. LA ARQUITECTURA WEB

Los sistemas típicos cliente/servidor pertenecen a la categoría de las aplicaciones de dos niveles. La aplicación reside en el cliente mientras que la base de datos se encuentra en el servidor. En este tipo de aplicaciones el peso del cálculo recae en los clientes, los cuales suelen ser máquinas menos potentes que los servidores. Además, está el problema de la actualización y el mantenimiento de las aplicaciones, ya que las modificaciones hechas deben ser trasladadas a todos los clientes[1].

Para solucionar estos problemas se ha desarrollado el concepto de arquitecturas de tres niveles: interface de presentación, lógica de la aplicación y datos.

La capa intermedia es invocada por el usuario para recuperar los datos deseados. La capa de presentación recibe los datos y los formatea para mostrarlos adecuadamente. Esta división entre la capa de presentación y la de la lógica permite una gran flexibilidad a la hora de construir aplicaciones, ya que se pueden tener múltiples interfaces de usuario sin cambiar la lógica de la aplicación. La tercera capa consiste en los datos que gestiona la aplicación. Estos datos pueden ser cualquier fuente de información como una base de datos o documentos XML[2].

[1] Borja Sotomayor. *Monografía de Aplicaciones Web*. Pág.: 3-5.
[2] Vegas, Jesús. 2002. *Creación de un Portal Web Docente*. Disponible en: <http://www.infor.uva.es>

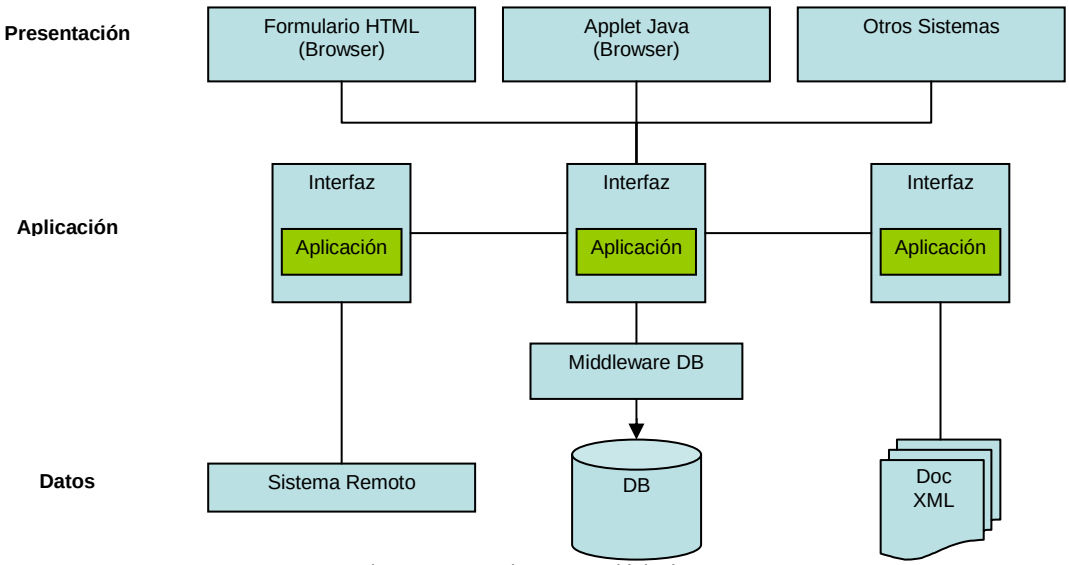


Figura 1.1: Arquitectura Multinivel
Fuente: www.infor.uva.es
Autor: Jesús Vegas

La gran mayoría de las aplicaciones Web suelen presentar este esquema de tres niveles. La capa de presentación incluye no sólo el navegador, sino también el servidor web que es el responsable de dar a los datos un formato adecuado.

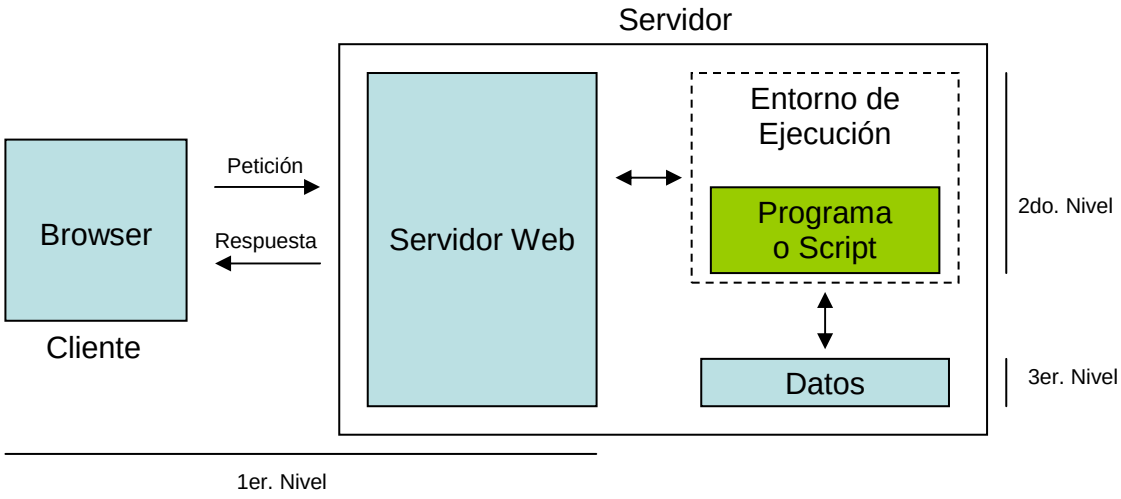


Figura 1.2: Arquitectura Web de tres niveles
Fuente: www.infor.uva.es
Autor: Jesús Vegas

El desarrollo de aplicaciones web se divide en básicamente en:

- Diseño gráfico (Web design): Se centra en la parte estética. Para esto se utilizan herramientas como Dreamweaver, Flash o Photoshop.

- Programación web (web programming): Se centra en el desarrollo de aplicaciones web. Entre las tecnologías más conocidas están: los lenguajes etiquetados (XHTML, XML), páginas activas (ASP, PHP, JSP), bases de datos (Oracle, SQL Server, MySQL).
- Ingeniería del software aplicada a la web (Web engineering)

Entra las aplicaciones web más comunes están:

- Sistemas de inscripción en línea
- Compras
- Banca en-línea
- Consultas
- Comercio electrónico, etc.

A continuación se presenta un cuadro comparativo entre las aplicaciones Cliente/Servidor y Web:

	Aplicación Cliente / Servidor	**Aplicación Web**
Aplicación Cliente	Software específico que el cliente debe tener instalado.	El navegador de internet. El cliente no tiene que instalar aplicaciones adicionales para ver la aplicación.
Interface	Parecido a cualquier aplicación del sistema. Posee ventanas, menús, barras, etc.	Páginas web programadas con formato HTML
Aplicación Servidor	Software específico que debe estar en el servidor.	Software específico que debe ser ejecutado por un servidor web.
Protocolo	Cualquier protocolo de comunicaciones. Generalmente se basa en el protocolo TCP/IP.	Protocolo HTTP.

Cuadro 1.1: Aplicaciones cliente/servidor vs. Aplicaciones web
Autor: Borja Sotomayor, 2003.

1.2.1. Los servidores web

Una página web puede visualizarse en cualquier computador sin necesidad de que esté conectada a internet. Para que las páginas web sean visibles se necesita un *Servidor Web* que es simplemente un computador que proporciona a los clientes documentos web como páginas HTML, junto con archivos de imagen, video, animación, texto, etc. En la computadora que se va a considerar como servidor se debe instalar el programa "servidor web" que no es más que una aplicación que atiende las peticiones de los navegadores web[3]. Los servidores web más conocidos en la actualidad son:

- Microsoft Internet Information Server (IIS) el cual corre únicamente en Windows.
- Apache que es un servidor web multiplataforma.

A continuación se presenta un gráfico que explica el proceso de comunicación que se da entre el cliente y el servidor en el entorno web:

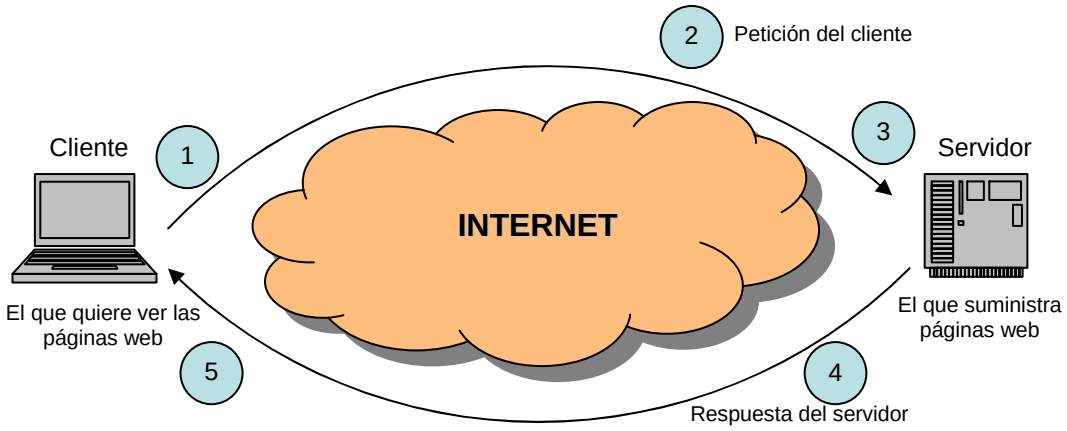


Figura 1.3: Como se comunican el cliente y el servidor
Fuente: Monografía de aplicaciones web, Pág.: 12
Autor: Borja Sotomayor

[3] Borja Sotomayor. *Monografía de Aplicaciones Web*. Pág.: 10-11.

1. El cliente hace el llamado a la página escribiendo el URL deseado
2. La petición del cliente es enviada por el browser
3. El servidor recibe la petición y la procesa
4. El servidor envía una respuesta al cliente
5. El cliente recibe el documente y lo procesa a través del browser

Las funciones que hace el servidor web son las siguientes:

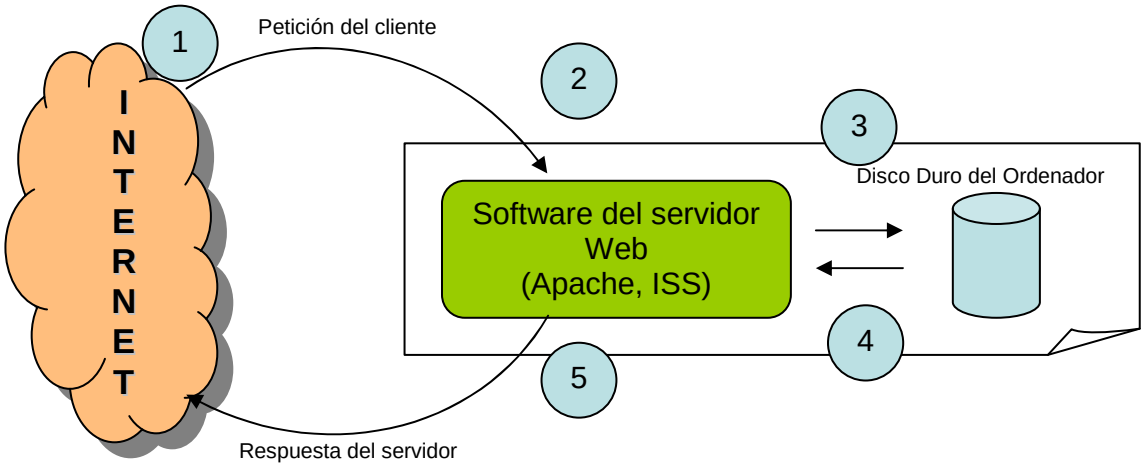


Figura 1.4: Las funciones del servidor
Fuente: Monografía de aplicaciones web, Pág.: 14.
Autor: Borja Sotomayor

1. Llega una petición del cliente
2. El servidor recibe la petición y la procesa, analiza la petición y decide la acción a tomar
3. Si la petición es valida, procede, accede al disco y recupera la información requerida
4. La información de respuesta es enviada al servidor
5. El servidor web envía la respuesta al cliente

1.2.2. Las páginas web dinámicas

Inicialmente las páginas web eran consideradas como estructuras estáticas en su contenido y tiempo. El contenido de las páginas permanecía constante y la información no se actualizaba con frecuencia. Al ser las páginas web estáticas, el servidor únicamente se limita a retransmitir los documentos web que se encuentran en su disco duro.

En la actualidad la información presentada y los procesos que se realizan cambian constantemente, por lo que la información debe manejarse de manera dinámica. En páginas dinámicas los contenidos son normalmente almacenados en una base de datos. Una base de datos es mucha más fácil de actualizar que un documento HTML.

La recuperación de información desde una base de datos a las páginas HTML se las hace mediante la utilización de *páginas activas*. Una página activa es un archivo de texto que contiene código HTML mezclado con código de programación. Una página activa ejecuta el código de programación y genera el código HTML necesario para que el browser pueda mostrar la información requerida[4]:

[4] Borja Sotomayor. *Monografía de Aplicaciones Web*. Pág.: 16-22.

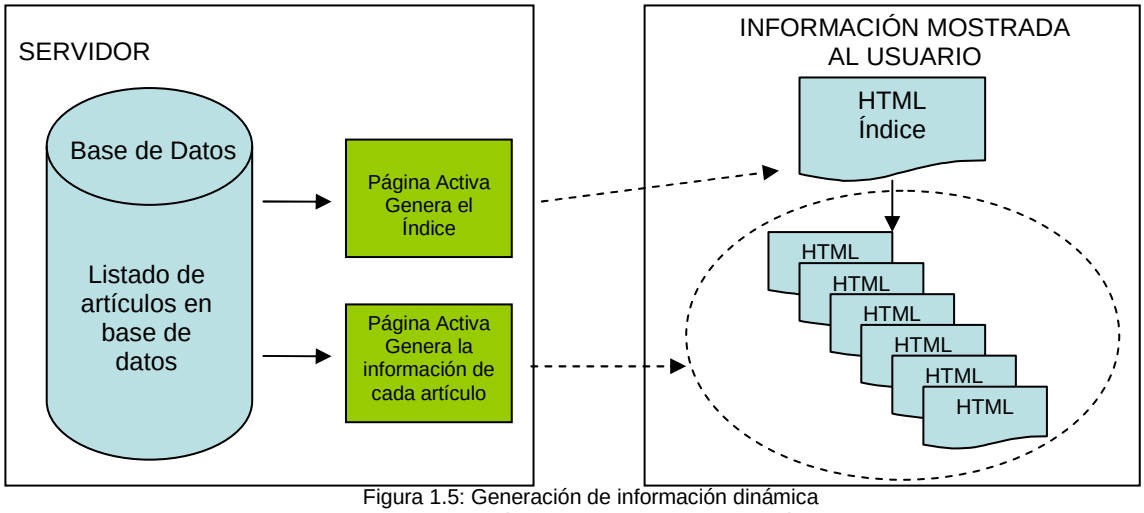


Figura 1.5: Generación de información dinámica
Fuente: Monografía sobre Aplicaciones Web, Pág.: 20
Autor: Borja Sotomayor

En este tipo de enfoque, el servidor desempeña una labor mucho más importante que en el enfoque estático, ya que el servidor debe ejecutar las páginas activas y acceder a la base de datos.

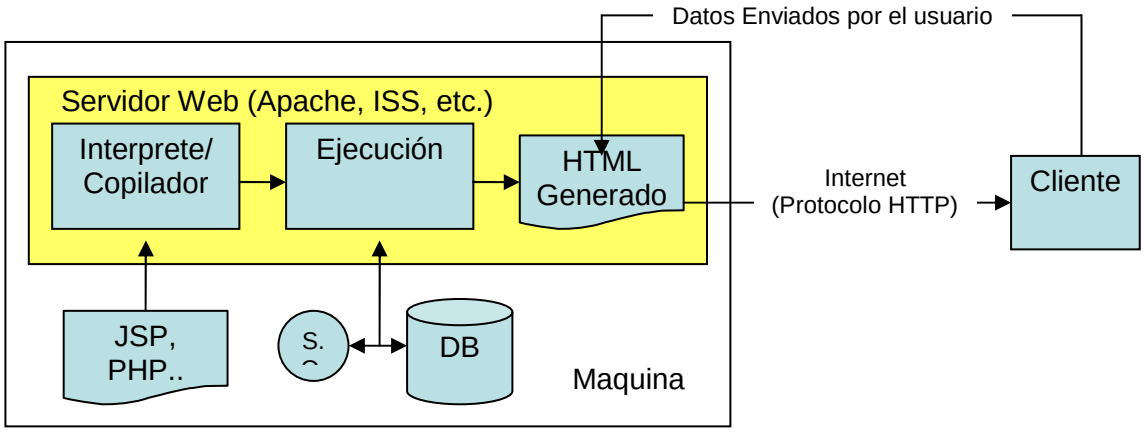


Figura 1.6: Arquitectura de generación de páginas dinámicas
Fuente: Monografía de aplicaciones web, Pág.: 22.
Autor: Borja Sotomayor

Existen mucha tecnologías de páginas activas, entre las que destacan ASP que trabaja bajo IIS en sistemas operativos Windows; PHP, JSP o *ColdFusion* que son multiplataforma, y no hay que olvidarse de CGI es considerado como el precursor se las páginas activas.

1.3. LA PLATAFORMA JAVA J2EE

1.3.1. Introducción al lenguaje Java

Para entender las características de la plataforma J2EE es necesario conocer un poco sobre el lenguaje de programación JAVA. Este lenguaje diseñado por la compañía Sun Microsystems Inc, con el propósito de crear un lenguaje que pudiera funcionar en redes computacionales heterogéneas (redes de computadoras formadas por más de un tipo de computadora, ya sean PC's, MAC's, etc.), y que fuera independiente de la plataforma en la que se vaya a ejecutar.

Las características principales de Java con respecto a cualquier otro lenguaje son:

Simple	Java elimina muchas de las características de otros lenguajes de programación como C++, para mantener reducidas las especificaciones del lenguaje y poder añadir otras características como el "garbage collector" (reciclador de memoria dinámica).
Orientado a objetos	Java implementa la tecnología Orientada a Objetos de C++. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo. Las plantillas de objetos son llamadas clases y sus copias son llamadas instancias. Estas instancias necesitan ser construidas y destruidas en espacios de memoria.
Distribuido	Java se ha construido con capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp lo que permite a los programadores acceder a la información a través de la red. Java en sí no es distribuido, pero proporciona las librerías y herramientas para que programas puedan ser distribuidos, y que se corran en varias máquinas.
Robusto	Java realiza verificaciones en busca de errores tanto en tiempo de compilación como en tiempo de ejecución. Esto ayuda a detectar errores en el ciclo de desarrollo. Java obliga a la declaración explícita de métodos para reducir las posibilidades de error. Maneja automáticamente la memoria para evitar corrupción de esta. Además, para asegurar el funcionamiento de la aplicación, realiza una verificación de los byte-codes que son el resultado de la compilación de un programa Java. Es un código de máquina virtual que es interpretado por el intérprete Java.
Arquitectura Neutral	El compilador Java compila su código en formato independiente de la arquitectura de la máquina en la que se ejecuta. Cualquier máquina que tenga el sistema de ejecución puede ejecutar ese código objeto, sin importar la máquina. En la actualidad se puede ejecutar aplicaciones Java en sistemas operativos Windows, Solaris, SunOs, GNU Linux, Irix, Aix, Mac de Apple, etc.

Seguro	Java tiene ciertas políticas que evitan se pueda codificar virus con este lenguaje. Se pueden habilitar muchas restricciones, especialmente para los applets, que limitan el acceso a los recursos críticos de una computadora. Java restringe el acceso a zonas delicadas de memoria o de sistema impidiendo la interacción con ciertos virus.
Interpretado	El intérprete Java (Java Run-Time) puede ejecutar directamente el código objeto. El código fuente escrito con cualquier editor se compila generando el byte-code. Este código intermedio es de muy bajo nivel, pero sin alcanzar las instrucciones de máquina propias de cada plataforma. El byte-code corresponde al 80% de las instrucciones de la aplicación. Este código se puede ejecutar sobre cualquier plataforma pero para ello hace falta el run-time, que es dependiente de la máquina y del sistema operativo. El run-time interpreta el byte-code y añade el 20% de instrucciones faltantes para su ejecución. Este sistema permite crear aplicaciones multiplataforma, pero se necesita el run-time correspondiente a cada sistema utilizado para ejecutarlas.
Multithreaded	Al ser multithreaded (multihilos), Java permite varias actividades simultáneas en un programa. Los threads son básicamente pequeños procesos o piezas independientes de un gran proceso. El beneficio consiste en un mejor rendimiento y mejor comportamiento en tiempo real.

Cuadro 1.2: Características de Java con respecto a otros lenguajes
Autor: Edison Hernández
Fuente: Tutorial de Java, Cap.: 1, Agustín Froufe.

Además como todo lenguaje orientado a objetos, el lenguaje Java implementa los siguientes conceptos:

Clases y objetos	Clase es un modelo abstracto de un tipo de objeto. Define sus métodos y atributos. Objeto es una instancia de una clase, es decir, la implementación con valores de un modelo abstracto. Las clases no son entidades independientes sino que se agrupan jerárquicamente heredando características y atributos. Cada instancia o implementación de una clase constituirá un nuevo objeto por lo que se pueden crear infinitos objetos distintos a partir de una sola clase.
Encapsulación	Se define como encapsulamiento a los datos ya que el acceso depende directamente de cada objeto. Permite abstraer los detalles internos de funcionamiento del objeto.
Herencia	Es el concepto que define la adopción de todas las características de una clase por parte de otra clase descendiente o heredera de la primera. La principal consecuencia de la herencia es la posibilidad de reutilizar clases, ya que se pueden crear nuevas a partir de las ya creadas. La herencia puede ser de dos tipos, simple si sólo es posible heredar características de una sola clase, o múltiple si se pueden heredar características de varias clases.
Polimorfismo	Generalmente el polimorfismo se entiende como la capacidad de tomar varias formas. Aplicado a los lenguajes de programación, debemos considerar dos tipos de polimorfismo: polimorfismo funcional el cual considera la posibilidad de referenciar, mediante el mismo identificador, diferentes funciones; y polimorfismo de datos que es la capacidad de un identificador para hacer referencia a instancias de distintas clases.

Cuadro 1.3: Conceptos de programación orientada a objetos
Autor: Edison Hernández
Fuente: Tutorial de Java, Cap.: 1, Agustín Froufe.

Para el desarrollo de aplicaciones en Java se necesitan de un paquete denominado SDK (Software Development Kit). La plataforma J2SE (Java 2 Standard Edition) trae dos productos principalmente: Java 2 Runtime Environment Standard Edition (JRE) y Java 2 Software Development Kit Standard Edition (SDK). Es la base de la tecnología Java pues incluye las herramientas de desarrollo, la máquina virtual JVM y la documentación. Es la tecnología básica para muchos estilos diferentes de desarrollo de software, incluidos applets, aplicaciones clientes y aplicaciones servidores individuales[5].

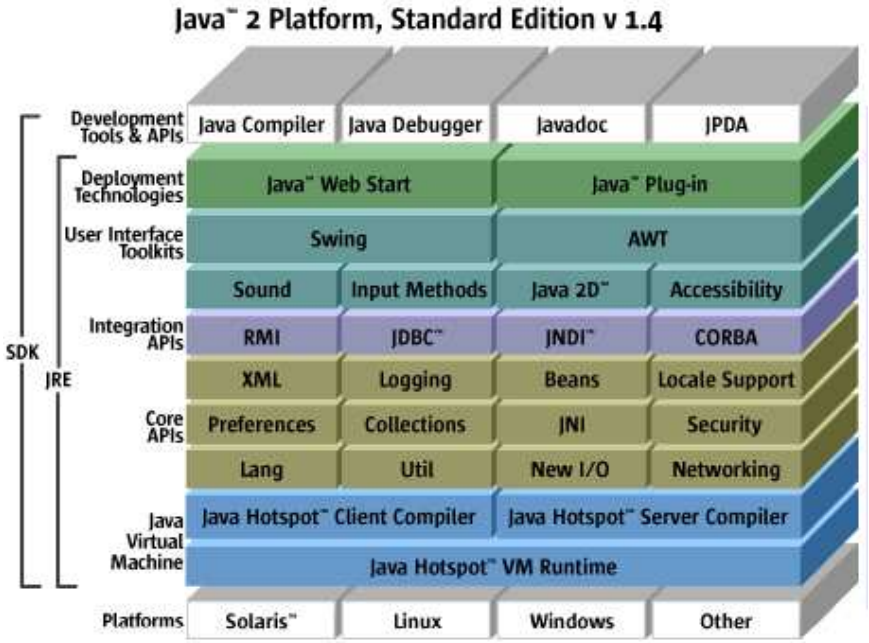


Figura 1.7: Componentes de la tecnología J2SE
Fuente: java.sun.com

J2SE es la base para construir la versión J2EE (Java 2 Enterprise Edition). J2EE es un conjunto de especificaciones que permiten el desarrollo, implementación y manejo de aplicaciones "multicapas", ya que agregan características de soporte para componentes como Java Servlets, JavaServer Pages, la tecnología XML y Enterprise JavaBeans.

[5] Sun Microsystems. *Java 2 Platform Standard Edition Overview*. Disponible en: <http://java.sun.com>

Existen otras versiones de paquetes Java disponibles: J2ME (Java 2 Platform, Micro Edition) es un paquete creado para el desarrollo de aplicaciones en dispositivos móviles como Palm's, teléfonos celulares o agendas electrónicas. J2RE (Java 2 Runtime Environment) ofrece la maquina virtual de Java (Java Virtual Machine) para ejecución de aplicaciones en el cliente. Es el componente de ejecución del Java Developer Kit (JDK) de Sun que permite la creación de applets Java[6].

1.3.2. Desarrollo de aplicaciones en J2EE

Java 2 Platform, Enterprise Edition (J2EE) define un estándar para el desarrollo de aplicaciones empresariales multicapa. Simplifica las aplicaciones empresariales basándolas en componentes modulares y estandarizados, proveyendo un completo conjunto de servicios, y manejando muchos de las funciones de la aplicación de forma automática, sin necesidad de una programación compleja[7].

Normalmente las llamadas aplicaciones empresariales se caracterizan por[8]:

- Acceso a bases de datos (BD): Normalmente relaciónales.
- Transaccionales: Propiedades ACID (Atomicity – Consistency – Isolation - Durability).
- Escalables: Deberían poder soportar más carga de trabajo sin necesidad de modificar el software (sólo añadir más máquinas).
- Seguras: No todos los usuarios pueden acceder a la misma informacion

[6] Sun Microsystems. *Java 2 Platform, Enterprise Edition (J2EE) Overview*. Disponible en: <http://java.sun.com/j2ee/overview.html>
[7] Sun Microsystems. *Java 2 Platform, Enterprise Edition (J2EE) Overview*. Disponible en: <http://java.sun.com/j2ee/overview.html>
[8] Borja Sotomayor. 2003. *Monografía Introductoria a J2EE*

- Tipo de interface: Entorno de ventanas utilizadas normalmente en intranets o páginas web en intranets y en Internet.
- Separación clara entre vista y modelo: Vista es el aspecto gráfico y el modelo es encapsula la lógica de negocio.
    - Ejemplo: aplicación bancaria donde el modelo es el conjunto de clases que permiten crear cuentas, eliminar, buscar, hacer transferencias bancarias, etc. La vista es la interface gráfica que permite a los empleados del banco realizar estas operaciones.
- Arquitecturas multicapa

Anteriormente en las aplicaciones cliente/servidor se presentaban muchas dificultades cuando se trataba de hacer cambios en la implementación de la capa de modelos. Para resolver estos problemas se hace uso de un servidor intermedio que contiene el modelo de negocio, entonces cuando se cambia la implementación del modelo, solo se afecta al servidor, los clientes solo presentan la interface gráfica y su función es acceder al servidor que implementa el modelo[9].

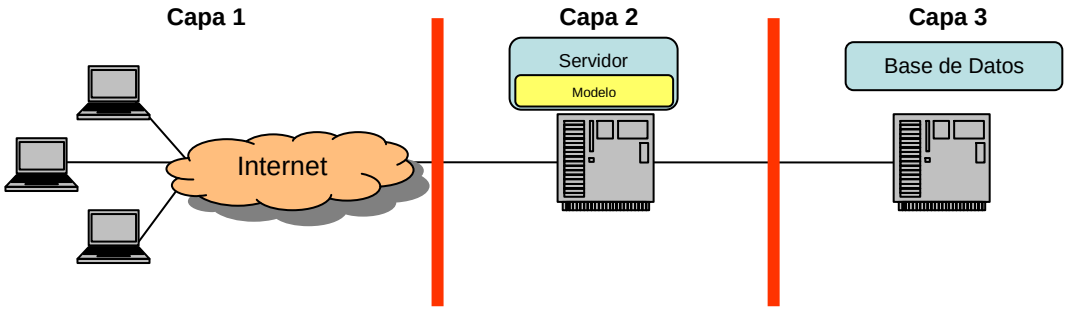


Figura 1.8: Aplicación Web 3 Capas
Fuente: Monografía Introductoria a J2EE
Autor: Borja Sotomayor

[9] Shannon, Bill. November 2003. *Java™ 2 Platform Enterprise Edition Specification*. Final Release

A diferencia de Microsoft .NET que es un producto, J2EE es un estándar, por lo que no es posible descargarlo y comenzar a desarrollar sino que es necesario usar alguna herramientas de desarrollo basadas en J2EE. Cada una estas herramientas proporcionan servicios adicionales a los existentes en la plataforma base[10].

J2EE toma ventaja de muchas características favorables de la versión *Stardard* (portabilidad de la aplicación, el *API JDBC* Java Data Base Connetion para acceso a base de datos; la tecnología *CORBA* para interacción de diferentes recursos en otros lenguajes o sistemas operativo) y un modelo de seguridad que protege la información en aplicaciones de internet. J2EE adicionalmente soporta componentes como Enterprise JavaBeans, Java Servlets, JavaServer Pages y XML entre otros[11].

Un *API (Application Program Interface)* es un método prescrito por el sistema operativo o por cualquier otra aplicación mediante el cual un programador que escribe una aplicación puede hacer solicitudes al sistema operativo.

Todas estas características permiten construir aplicaciones con arquitecturas *multicapas*. En Ingeniería del Software, una arquitectura multicapas describe el grado de separación que existe entre los diferentes componentes de software para facilitar los procesos de una aplicación. Por ejemplo la capa intermedia en una aplicación permite hacer más eficiente la interacción entre el usuario y la base de

[10] Sheil, Humphrey; Monteiro Michael. *Rumble in the jungle: J2EE versus .Net, Part 1*. Disponible en: <http://www.javaworld.com>
[11] Sun Microsystems. *Java 2 Platform, Enterprise Edition (J2EE) Overview*. Disponible en: <http://java.sun.com/j2ee/overview.html>

datos, y posibilita tener diferentes interfaces de usuario en la misma aplicación. La arquitectura multicapas más utilizada es la arquitectura de tres capas.

1.3.3. Características

J2EE ofrece las siguientes características:

Manejo de Excepciones	Indican cuando se presentan errores inesperados de ejecución. Ayudan a la detección más temprana de errores y a la corrección de estos.
Loggin	J2EE viene incluido con librerías para control de acceso a páginas seguras.
Configuración	Componentes del lado del servidor como Servlets o EJBs pueden contener información de configuración a través de archivos XML o archivos planos.
Autentificación	Existen componentes destinados al manejo de autentificaciones para usuarios y roles. Simplemente se indica que recursos deben ser protegidos.
Autorización	J2EE soporta especificaciones de seguridad a través de configuraciones en archivos planos o XML.
Manejo de sesiones	Los Servlets y JSPs permiten el manejo de sesiones utilizando las conocidas *cookies* o también clases propias de Java como por ejemplo la clase *HttpSession*

Cuadro 1.4: Características de la plataforma J2EE
Fuente: Sheil, Humphrey & Monteiro Michael; Rumble in the jungle: J2EE versus .Net, Part 2
Autor: Edison Hernández

1.3.4. Arquitectura

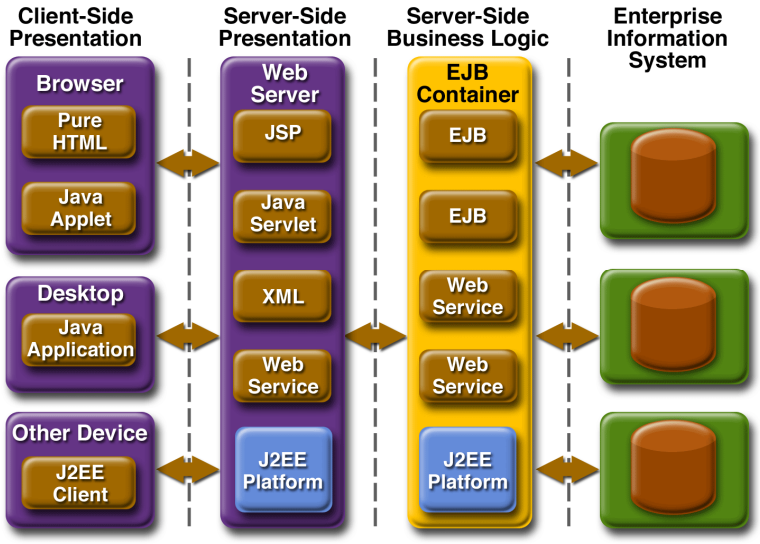


Figura 1.9: Arquitectura de la tecnología J2EE
Fuente: Java 2 Platform, Enterprise Edition (J2EE) Overview
Autor: Sun Microsystems

J2EE es un conjunto de especificaciones de programas (APIs – Application Program Interface) Java para la construcción de aplicaciones empresariales donde la mayor parte de las abstracciones de las APIs corresponden a interfaces y clases abstractas. La arquitectura de J2EE se puede dividir en cinco partes[12]:

- El lenguaje de programación java
- El modelo de programación del cliente
- La infraestructura de la capa de middleware
- La API de negocios para los programadores
- La API no visible para los programadores.

J2EE permite el desarrollo de aplicaciones multicapas por lo que se encapsula cada capa en tipos específicos de componentes: La lógica de negocios puede ser encapsulada dentro de clases Java, La interacción del cliente con la aplicación se puede dar a través de páginas planas HTML, applets, Java Servlets, páginas dinámicas JSP, o incluso a través de aplicaciones Java "stand-alone" si se quiere manejar la estructura cliente/servidor estándar.

Gracias a este tipo de arquitectura, varios componentes J2EE pueden ser reutilizados, lo cual es de gran ayuda para aplicaciones corporativas. J2EE presenta componentes predefinidos para autentificación o seguridad, pero además permite que estos componentes puedan ser creados y modificados a partir de los ya existentes.

El modelo de aplicaciones J2EE divide las aplicaciones en 3 partes fundamentales: los componentes, los contenedores y los conectores:

[12] Sun Microsystems. *Java 2 Platform, Enterprise Edition (J2EE) Overview*. Disponible en: <http://java.sun.com/j2ee/overview.html>

Los componentes	Se refieren a todas las tecnologías de desarrollo utilizadas para el funcionamiento de la aplicación en sus diferentes capas (JSP, Servlets, EJB, etc.)
Los contenedores	Interactúan entre los componentes y los clientes, proveyendo servicios transparentes para ambos. Incluye soporte transaccional y manejo de recursos.
Los conectores	Se encuentran por debajo de la plataforma J2EE. Definen APIs que comunican los diferentes componentes de la aplicación. Los conectores implementan plug-ins de mensajería bidireccional para comunicación entre los componentes y sistemas.

Cuadro 1.5: Partes de una aplicación J2EE
Fuente: Borja Sotomayor; Monografía Introductoria a J2EE
Autor: Edison Hernández

Los contenedores proveen de un ambiente estándar de ejecución para los componentes. El ambiente de ejecución provisto por los contenedores debe ser compatible con el definido en la especificación J2SE. Los componentes pueden asumir que el contenedor, junto a sus servicios y APIs, van a estar disponibles en cualquier implementación de la plataforma J2EE.

Los contendores especifican un conjunto de APIs que los componentes deben implementar. Dichas APIs definen el mecanismo de comunicación entre el contenedor y el componente. Las APIs de servicios proveen acceso a las extensiones estándar definidas por J2EE (por ejemplo JDBC)[13].

1.3.4.1. Componentes

J2EE provee las herramientas necesarias para construir aplicaciones empresariales en corto tiempo, con capacidades de seguridad, estabilidad y mantenimiento:

[13] Sun Developer Network Community. *J2EE Connector Architecture*

JavaServer Pages (JSPs)	Genera contenido dinámica para navegadores web y dispositivos móviles. JSP permite la generación de código HTML mediante la utilización de tags (etiquetas) personalizadas, esto evita escribir código Java dentro de la página y la vuelve más fácil de mantener.
JavaServer Faces	Es un marco de trabajo de interfaces de usuario del lado de servidor para aplicaciones Web.
Java Servlets	Permite construir la lógica de navegación y de control en aplicaciones J2EE siguiendo el patrón de diseño Model-View-Controller (MVC) en conjunto con páginas JSP.
Enterprise JavaBeans (EJBs)	Definen la lógica de negocio de la aplicación. Existen 2 tipos principales de EJB: *Session Beans (Beans de sesión)* para modelar la lógica del negocio, permiten implementar fachadas del modelo, se definen con interfaces remotas para separación física con la interface gráfica. *Entity Beans (Beans de entidad)* para modelar la persistencia de datos. Proveen gran escalabilidad y un framework muy seguro. Permiten implementar los objetos persistentes del modelo, permitiendo construir una capa que no depende de un tipo particular de BD.
Web services	Los servicios web son un conjunto de tecnologías que usan XML para intercambio de información en un entorno distribuido. Estos servicios web al ser APIs permiten ser utilizados junto con varios lenguajes. Son considerados una buena solución para integración de aplicaciones en Internet, ya que todos los firewalls los reconocen con el protocolo http y todos los fabricantes de tecnología proporcionan soporte para Servicios Web
Java Connectivity Architecture (JCA)	Permite a las aplicaciones empresariales interactuar con aplicaciones "no-Java".
Java Message Service (JMS)	Provee capacidad de mensajera asincrónica a la plataforma J2EE. Establece una capa intermedia orientada a menajes para hacer la comunicación en aplicaciones multicapas más fácil.
Java Management Extensions (JMX)	Maneja servidores y aplicaciones J2EE.
Java Naming and Directory Interface (JNDI)	Un componente requerido de la plataforma para manejo de directorios y nombres de archivos.
JavaMail	API estándar independiente del protocolo que permite el envío y la recepción de correos.
Java Transaction API ((JTA)	API de alto nivel que permite controlar Transacciones Distribuidas. Permite crear interfaces entre para manejo de transacciones en sistemas distribuidos.
Java API for XML Processing (JAXP)	Provee acceso a APIs estándares para que una aplicación pueda transformar documentos XML independientemente de la implementación manejada.

Cuadro 1.6: Componentes de la plataforma J2EE
Fuente: Sun Microsystems; Sun ONE Architecture Guide, Cap.: 3 Components and Containers
Autor: Edison Hernández

Además de estos componentes, utiliza otros que son parte de la plataforma J2SE como:

Java Database Connectivity (JDBC)	API para acceso a bases de datos relacionales, trabajan como conectores para cualquier tipo de base de datos. El programador interactuar con la base de datos mediante estos conectores y el uso de queries (consulta,

	actualización, inserción y borrado).
JavaBeans	Un JavaBean o bean es un componente hecho en software que se puede reutilizar y que puede ser manipulado visualmente por una herramienta de programación en lenguaje Java.
HotSpot Virtual Machine	Permite correr aplicaciones en modo de servicio por lo que se tiene un mejor manejo de memoria
The Java Virtual Machine	Para ejecución de aplicaciones Java del lado del cliente

Cuadro 1.7: Componentes de la plataforma J2SE
Fuente: Sun Microsystems; Sun ONE Architecture Guide, Cap.: 3 Components and Containers
Autor: Edison Hernández

En el siguiente gráfico se puede ver su estructura:

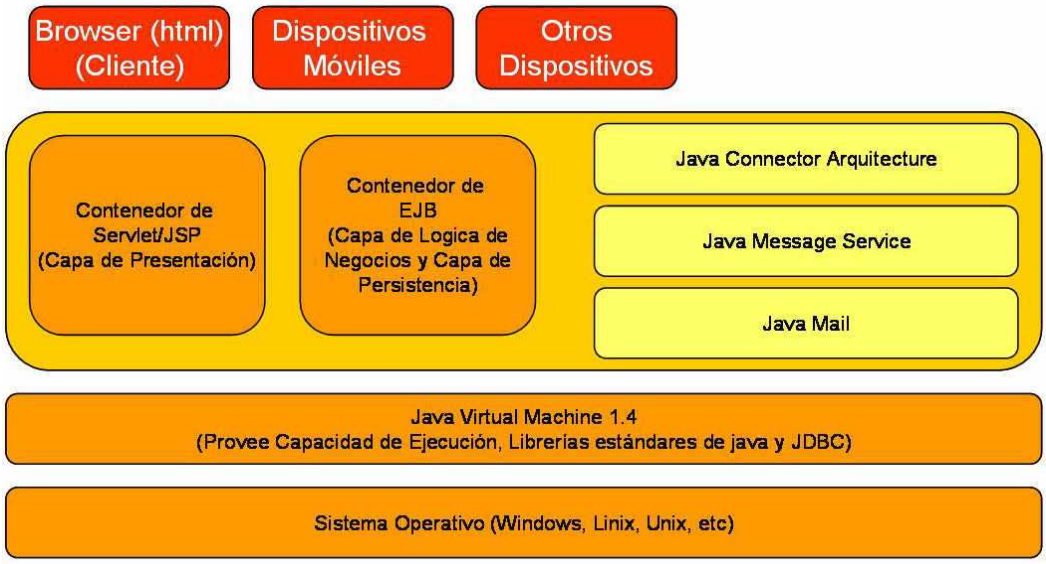


Figura 1.10: La Plataforma J2EE
Fuente: Java 2 Platform, Enterprise Edition (J2EE) Tutorial
Autor: Sun Microsystems

1.3.5. Requerimientos

Los requerimientos de la plataforma J2EE se basan en las necesidades de Java SDK (Software Development Kit).

1.3.5.1. Requerimientos de Hardware

Los requerimientos específicos de hardware para desarrollo de aplicaciones J2EE son difíciles de determinar dada la gran variedad de configuraciones posibles. Los

requerimientos también varían de acuerdo al número de usuarios concurrentes que acceden al servidor:

Procesador	PC con procesador Pentium III de 500 MHz o superior
Memoria	256 Mbytes de RAM
Disco Duro	Disco duro con 200 Mbytes de espacio libre
Otros Dispositivos	Tarjeta de red o en su defecto Modem Tarjeta de video SVGA

Cuadro 1.8: Requerimientos de hardware para desarrollo
Fuente: JavaTM 2 SDK, Standard Edition Installation Instruction <http://java.sun.com/j2se>

Para el cliente se requieren configuraciones básicas:

Procesador	PC con procesador 586 o superior
Memoria	64 Mbytes de RAM
Disco Duro	Disco duro con 30 Mbytes de espacio libre

Cuadro 1.9: Requerimientos de hardware para clientes
Fuente: JavaTM 2 SDK, Standard Edition Installation Instruction <http://java.sun.com/j2se>

1.3.5.2. Requerimientos de Software

Los requerimientos indicados en esta sección se los hace de acuerdo a las especificaciones del sistema operativo Windows que se usará como plataforma para la ejecución de este estudio.

Sistema Operativos	Windows 98, NT, Me, 2000 o XP
Navegador de internet	Cualquier estándar
Paquete de desarrollo	Paquete SDK de Java 1.4.1

Cuadro 1.10: Requerimientos de software
Fuente: JavaTM 2 SDK, Standard Edition Installation Instruction <http://java.sun.com/j2se>

1.4. DESARROLLO DE APLICACIONES PARA INTERNET EN JAVA

En esta sección nos enfocaremos en los componentes de la plataforma J2EE que son más utilizados para el desarrollo de aplicaciones para internet

1.4.1. Servlets

El Servlet se considera una evolución de los CGIs. Fue desarrollada por SUN Microsystems como parte de la tecnología JAVA. Son aplicaciones que se ejecutan en el motor de Servlets (*Servlet engine*) el cual es parte del servidor web[14].

Inicialmente era considerada una herramienta muy importante a la hora de desarrollar aplicaciones para internet, pero en la actualidad los Servlets están en desuso, ya que han evolucionado hacia los JSP's.

Los Servlets aportan una forma de comunicación entre el servidor y el cliente. Poseen un modelo general de clases para ejecutar distintos servicios. Tanto los API's del los Servlets como el motor de Servlets son incluidos dentro del paquete *j2sdk* de Sun Microsystems.

Los Servlets utilizan un *contenedor de Servlets* el cual se encarga, entre otras cosas, de pasar las peticiones del cliente al Servlet en el servidor y por último devolver la respuesta solicitada de nuevo al cliente. La implementación actual del Servlet difiere

[14] Hall, Marty. *Servlets y JSP*. Disponible en: <http://www.programacion.com/java/tutorial/servlets_jsp/>

de un programa a otro, pero la interface entre el contenedor y el servlet se especifica en el API del servlet.

Básicamente, el *ciclo de vida* de un servlet es administrado por el *contenedor de Servlets* mediante tres métodos definidos en la interface Servlet: *init*, *service* y *destroy*:

Un servlet es construido e inicializado, después se procesan cero o varias peticiones y por último se destruye. El servlet es cargado una sola vez y está residente en memoria mientras se procesan las respuestas. La interface que define esta estructura se llama *javax.servlet.Serlvet*.

En el método *Init()* se empieza la vida de un servlet, es llamado inmediatamente después de ser instanciado y es llamado una sola vez. El método *Init()* crea e inicializa los recursos que serán usados mientras se manejan las peticiones.

El método *Service()* maneja las peticiones enviadas por el cliente. No pueden dar comienzo a los servicios de las peticiones hasta que el método *Init()* se ejecute. La implementación más habitual del método *Service()* está en la clase *HttServlet*.

El método *Destroy()* significa el final de la vida de un Servlet. Cuando un servicio se finaliza se llama al método *Destroy()*. Este método 'limpia' todos los recursos creados en el método *Init()*. Aquí es el lugar dónde debe cerrarse conexiones

existentes a base de datos, también es el lugar dónde se debe guardar información persistente en caso de que se la utilice en algún otro servicio[15].

1.4.2. JavaServer Pages (JSPs)

Las JavaServer Pages (JSPs) son una de las tecnologías que han extendiendo la funcionalidad de los servidores web.

Las páginas JSP sirven para crear páginas web dinámicas y se basan en un archivo de texto que contiene código HTML o XML junto con elementos Java. Cuando un cliente hace una petición a una página JSP del sitio web que no se ha ejecutado antes, la página es inicialmente pasada al motor de JSP's que compila la página convirtiéndola en servlet, después la ejecuta y devuelve el contenido de los resultados al cliente en un formato entendible por el browser[16].

Es posible ver el código del servlet generado el cual contiene las clases *JSPPage* y *HttpJspPage*. Estas clases definen la interface para el compilador de páginas JSP. Entre los *elementos* de una página JSP podemos encontrar:

Directivas	Dan información global de la página, por ejemplo, importación de librerias o página para manejo los errores. Su sintaxis general es <%@ directiva {atributo ="valor"} %> dónde la directiva debe tener un número de atributos. Las posibles directivas en JSP son: • Page: Información para la página. Tiene varios atributos o language="java": Comunica al servidor el lenguaje que va a ser utilizado en el archivo. *java* es el único valor posible. extends="package.class": Define la clase padre del servlet generado. o import="package.*,package.class": Sirve para especificar los paquetes y clases que se requieren en la ejecución. o session="true|false": Por defecto *session* es *true*, manteniendo

[15] Juan Antonio Palos. *Tutorial de Servlets (Básico)*. Disponible en: <http://www.programacion.com/java/tutorial/servlets_basico/>
[16] Hall, Marty. *Servlets y JSP*. Disponible en: <http://www.programacion.com/java/tutorial/servlets_jsp/>

	los datos de la sesión para la página. o isThreadSafe="true|false": Por defecto es *true*, le hace señales al motor de JSP para que múltiples pedidos del cliente puedan ser tomadas como una. o info="text": Información en la página a la que puede accederse a través del método *Servlet.getServletInfo()* o errorPage="página_error": Página que manejará los errores. o isErrorPage="true|false": Indica si la página manejará los errores. • Include: Incluye archivos completos palabra por palabra. • Taglib: La dirección de la librería de tags que se usará en la página.
Declaraciones	Sirven para declarar métodos y variables. Una declaración de JSP puede definirse como una definición de variables y métodos que son usadas en la página. Un bloque de declaraciones sería: <%! declaración %>. Ej: <%! String strCadena = "x"; %>
Scripts de JSP	Los Scripts son bloques de código Java residentes entre los tags <% y %>. Estos bloques de código están dentro del Servlet generado. Los Scripts pueden acceder a cualquier variable o método declarado.
Expresiones de JSP	Formatea las expresiones como cadenas para incluirlas en la página de salida. Las expresiones son una herramienta para insertar código embebido dentro de la página HTML. Cualquier cosa que esté entre los tags <%= y %> será evaluado, convertido a cadena y posteriormente mostrado en pantalla.

Cuadro 1.11: Elementos de una página JSP
Fuente: Eric Armstrong, Jennifer Ball; The J2EE™ 1.4 Tutorial; August, 2004
Autor: Edison Hernández

1.4.3. Java Beans

Cuando analizamos el desarrollo de la arquitectura de una aplicación que envuelve un JSP, es una buena idea intentar poner toda la lógica de negocio en componentes reutilizables. Estos componentes pueden ser insertados dentro de la página cuando son requeridos mediante los llamados *JavaBeans* que son simplemente clases Java que se adapta a los siguientes criterios:

- Clase publica.
- Constructor público sin argumentos
- Posee métodos públicos "Set" y "Get".

Las propiedades son siempre colocadas y recuperadas utilizando una convención. Para cada propiedad, deben existir dos métodos, uno getxxx() y otro setxxx() dónde xxx es el nombre de la propiedad.

Casi en todos los casos los Beans son usados para encapsular elementos de la interface gráfica de usuario sean estos visuales y no visuales. Actualmente se piensa en ellos como componentes que simplemente encapsulación de código Java que pueden ser unirse visualmente en componentes compuestos, applets, aplicaciones y servlets utilizando herramientas visuales de desarrollo de aplicaciones [17].

1.4.4. Enterprise JavaBeans (EJB's)

Enterprise Java Beans (EJB's) son utilizados para separar la lógica del negocio de la lógica de presentación y así evitar que se las mezcle en los Servlets y páginas JSP.

Existen 3 tipos de EJB's:

Session Beans	Modelan procesos de negocios, y pueden hacer cualquier cálculo necesario o acceder a otros sistemas. La diferencia fundamental de los Session Beans con los Entity Beans es en el tiempo de vida, ya que los primeros solo duran lo que dura la sesión del cliente mientras que los segundos permanecen durante el tiempo de vida de la aplicación. Existen de 2 tipos: • *Stateful Session Beans*: El estado conversacional se mantiene mientras la sesión con el cliente esté viva. Se utilizan para representar procesos de negocio que se expanden a través de varias invocaciones a métodos. Un ejemplo de esto sería el carrito de compras del Web en una tienda virtual. • *Stateless Session Beans*: El estado conversacional se mantiene por el tiempo que dura la invocación a un método. Luego de cada invocación el contenedor puede optar por destruir el Bean, recrearlo o inicializarlo. Estos normalmente deben recibir del cliente toda la información necesaria para realizar su tarea, o en todo caso obtenerla de una fuente externa como una base de datos.
Entity Bean	Modelan la información del negocio. Son entidades persistentes con respecto a la base datos, ya que permanecen y almacenan en memoria los datos durante una transacción. Pueden compartirse por múltiples clientes y un cliente puede

[17] Sun Microsystems. *Tutorial de Beans (Básico)*. Disponible en: <http://www.programacion.com/java/tutorial/beans/>

	pasar el objeto por referencia a otro. El estado se cambia transaccionalmente; además se recupera frente a fallas; y un cliente puede seguir utilizando la referencia al objeto después que el container lo re-inicializa luego de la falla. Un Entity Bean por ejemplo: podría representar una línea de una factura, una factura entera o todas las facturas realizadas en un día. Existen 2 tipos de Entity Beans: • *Bean Managed Persistent (BMP)*: el desarrollador del Bean es responsable de escribir el código para persistirlo. • *Container Managed Persistent (CMP)*: el contenedor es responsable de manejar la persistencia.
Message Driven Beans	Son parecidos a los Session Beans pero enfocados a atender invocaciones por mensajes. Permiten incluir mensajería asincrónica. Son *stateless*, es decir no tienen un estado conversacional. Son un tipo especial de EJB que puede recibir mensajes generados por un cliente.

Cuadro 1.12: Tipos de EJBs
Fuente: Sun Microsystems; Sun ONE Architecture Guide; Cap.: 3, J2EE™ Components and Containers
Autor: Edison Hernández

Si se realizaran transacciones bancarias se utilizaría un *Session Bean* para representar un retiro de un cajero automático, e invocar a un *Entity Bean* que representa la cuenta. Hay que tener en cuenta que se tienen invocaciones a Session Beans y a Message Driven Beans pero no a Entity Beans.

Dentro de la arquitectura de J2EE podemos ver un mapeo de capas de la tecnología Enterprise Java Beans:

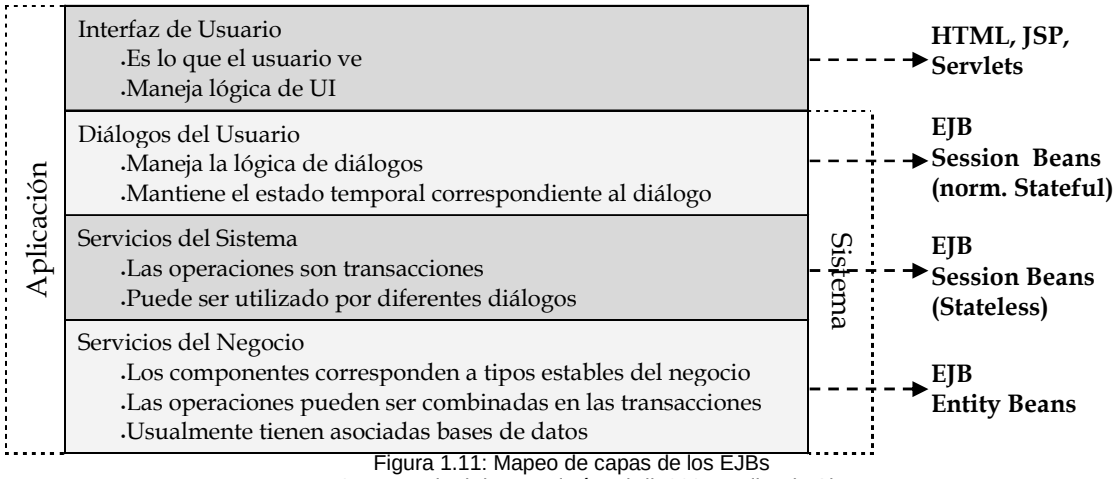


Figura 1.11: Mapeo de capas de los EJBs
Fuente: InCo – Facultad de Ingeniería; Abril, 2004; Taller de Sistemas de Información 1

Los EJBs manejan tres formas de marcar las transacciones:

- *Programáticamente (Bean Managed Transaction)*: En el propio código de los componentes se inician y terminan las transacciones Los componentes deben manejar los métodos *begin*, *commit* y *rollback*.
- *Declarativamente (Container Managed Transaction)*: Se maneja especificando los ejes para cada método de un componente, luego el contenedor se encarga de que esto sea así; sin que los componentes tenga nunca que hacer *begin*, *commit* o *rollback*.
- *Por el cliente*: El cliente del Bean es el que controla la transacción, la inicia antes de realizar las invocaciones y la termina cuando éstas finalizan.

1.4.5. Java Database Connectivity (JDBC)

JDBC es una API puro de Java que se usa para ejecutar comandos de SQL. Suministra una serie de clases e interfaces que permiten al desarrollador escribir aplicaciones que gestionen bases de datos.

La interacción típica con una base de datos consta de los siguientes cuatro pasos:

- Abrir la conexión a la base de datos
- Ejecutar consultas contra la base de datos
- Procesar los resultados
- Cerrar la conexión

Lo primero que se debe hacer para poder "conectar" bases de datos con páginas JSP o con Servlets es instalar el *driver* correcto de la base de datos que se va a utilizar.

Actualmente hay JDBCs para casi todas las bases de datos disponibles en el mercado entre las que resaltan MySQL, Sybase, Oracle o SQLServer[18].

[18] Sun Microsystems. *JDBC Overview*. Disponible en: <http://java.sun.com/products/jdbc/overview.html>

Capítulo 2

2. IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)

2.1. CONCEPTOS BÁSICOS

En la actualidad se considera que los desarrolladores de software se dividen en dos tipos: los que usan entornos de desarrollo integrado o IDEs (Integrated Development Environments) y los que no. Estos últimos prefieren un editor de texto (como wordpad o el Bloc de notas), un compilador y un depurador. Los pertenecientes al primer tipo prefieren usar IDEs como ayuda para generación del código y construcción de proyectos.

No hace mucho la única manera de desarrollar aplicaciones en C, COBOL o Fortran era recurrir a un editor de texto, un compilador y un depurador. Con la llegada de los lenguajes de cuarta generación comenzaron a aparecer las primeras herramientas de desarrollo integrado, muy primitivas comparadas con las que podemos encontrar ahora, ya sean gratuitas o comerciales.

Cualquier entorno actual de desarrollo integrado ofrece al menos control del editor de código, del compilador y del depurador desde una única interface de usuario. Su misión consiste en evitar tareas repetitivas, facilitar la escritura de código correcto, disminuir el tiempo de depuración e incrementar la productividad del desarrollador. Estas tareas pueden realizarse de muchas maneras distintas, ya sea mediante la inclusión de asistentes para las tareas más habituales y mecánicas, o el uso de

editores que completen automáticamente el código y señalen errores sintácticos o de otro tipo.

En teoría, un entorno de desarrollo integrado o IDE debería aportar funcionalidades al desarrollador durante todas las etapas del ciclo de vida del desarrollo de software (desde el análisis y diseño, a la distribución del producto, y su mantenimiento); de ahí la palabra "integrado". En la práctica, solamente los IDEs más modernos cumplen esta condición y, a menudo, de forma incompleta.

Después del éxito de Visual Basic, ha sido frecuente asumir que los IDEs necesitan incluir herramientas visuales de generación de interfaces de usuario (GUI builders), pero esta premisa resulta inexacta. Algunos IDEs carecen de diseñadores gráficos visuales y no por ello dejan de ser excelentes. En el caso específico de Java, la mayor parte de las aplicaciones actuales se ejecutan en el lado del servidor y no precisan de interface gráfico. Muchas veces resulta más conveniente disponer de un buen editor JSP/HTML que de un vistoso diseñador gráfico.

Los IDEs también tienen sus desventajas. En comparación con el clásico modo de programar (editor de texto-compilador-depurador), los IDE’s consumen muchos más recursos, tienden a ser lentos (particularmente los escritos en Java) y su manejo requiere un cierto aprendizaje. Algunos IDEs son sumamente complejos de manejar, incluso para llevar a cabo las tareas aparentemente más sencillas y, en ocasiones. los manuales no ofrecen la guía esperada.

Todavía no existe un IDE universal o perfecto, capaz de reunir todas las características que un desarrollador puede necesitar. Por lo general, los puntos débiles de un IDE coinciden con los puntos fuertes de otro. Por esto el desarrollador que trabaje en varios campos simultáneamente (desarrollo de servicios Web, creación de clases, diseño de páginas web, edición de XML) necesitará usar varios IDEs u otras herramientas para trabajar de forma óptima.

En algunas ocasiones, la elección del IDE por parte de los desarrolladores no es libre, sino que está condicionada por varios aspectos (plataforma, bases de datos, servidores de aplicaciones). Existen empresas que suministran componentes o módulos, añadidos ya a sus herramientas o por separado, con el fin de proporcionar soporte para las plataformas J2EE más populares. Borland proporciona, por ejemplo, módulos en su JBuilder para WebLogic, Tomcat, iPlanet (Sun ONE) y WebSphere, pero WebSphere y WebLogic (de IBM y BEA, respectivamente) no suministran directamente módulos para JBuilder porque son los líderes en servidores de aplicaciones y prefieren dirigir a los desarrolladores hacia sus propios productos.

Normalmente dentro de los IDEs comerciales, del 80% de las características incorporadas sólo son útiles para el 20% de los usuarios. Muchas veces se encuentra IDEs poco considerados que exprimen las capacidades de las maquinas. Muchas veces el abuso de los recursos del sistema se origina por la instalación del IDE con muchas características poco o nada utilizadas. Esta inclusión de utilidades no solicitadas (ni necesitadas) también redunda en el precio del producto.

Pese a todos estos inconvenientes, los IDEs suelen proporcionar una importante ayuda a la hora de conservar un registro de las versiones, generar y mantener la documentación del proyecto, y evitar tareas monótonas y repetitivas[19].

El precio también puede interferir en la elección de una herramienta como esta, pero gracias al auge de aplicaciones *Open Source* se puede encontrar gran calidad sin tener que pagar grandes sumas de dinero. *Open Source* es un término por el que se conoce al software distribuido y desarrollado libremente. Este tipo de licencia además específica que los códigos fuentes de un programa pueden estar a disposición del público de manera parcial o total, y que estos pueden ser modificados y adecuados a las necesidades de cada usuario[20].

[19] Abián, Miguel Ángel. 2003. *El Archipiélago Eclipse*. Disponible en: <http://www.javahispano.org/articles.article.action?id=75>
[20] Wikipedia. *Definición de Código Abierto*. Disponible en: <http://es.wikipedia.org/wiki/C%C3%B3digo_abierto>

2.2. PRINCIPALES PROPUESTAS DEL MERCADO

2.2.1. Eclipse

2.2.1.1. Historia

Eclipse es un IDE *Open Source* y extensible. Esto quiere decir que es gratuito y permite agregar extensiones adicionales para mejorar su funcionalidad.

Eclipse es parte de un proyecto que se lanzó originalmente en Noviembre de 2001, cuando IBM donó 40 millones de dólares del código fuente de WebSphere Studio Workbench y formó el Eclipse Consortium con empresas fabricantes de herramientas de software; todo esto para controlar el desarrollo continuado de la herramienta.

Originalmente la junta directiva del consorcio incluía a empresas como Borland, MERANT, IBM, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft y Webgain; en junio de 2002 se agregaron Hitachi, Instantiations, MontaVista, Scapa Technologies, Telelogic, Trans-Enterprise Integration y Serena; en septiembre de 2002 se sumaron ETRI, HP, MKS, SlickEdit y se aprobó la entrada de Oracle; en diciembre de 2002 entraron como miembros AltoWeb, Catalyst Systems, Flashline, Parasoft, SAP, teamstudio y TimeSys. El grupo OMG (Object Management Group) también forma parte de la junta directiva[21].

[21] Abián, Miguel Ángel. 2003. El Archipiélago Eclipse. Disponible en: <http://www.javahispano.org/articles.article.action?id=75>

Inicialmente, el *Eclipse Consortium* se había enfocado en tres proyectos principales:

- *The Eclipse Project* responsable de desarrollar el workbench (banco de trabajo) del IDE Eclipse, el JDT (Java Development Tools), y el PDE (Plug-In Development Environment) utilizado para ampliar la plataforma.
- *The Eclipse Tools Project* enfocado en la creación de herramientas para la plataforma Eclipse. Entre los subproyectos actuales se incluyen un IDE Cobol o un IDE C/C.
- *The Eclipse Technology Project* se enfoca en investigaciones tecnológicas y de educación utilizando la plataforma Eclipse.

2.2.1.2. Tipo de Licencia

Eclipse se distribuye actualmente bajo la licencia CPL (Common Public License) versión 1.0 de IBM, aprobada por la organización *Open Source Initiative* (OSI). A diferencia de otros proyectos *Open Source* que no permiten que se deriven de ellos trabajos propietarios o cerrados, Eclipse puede extenderse mediante la inclusión de plug-ins propietarios o ser usado como base para la creación de nuevas herramientas y, tras reempaquetarse y compilarse el código resultante, el producto final puede venderse de forma comercial, manteniéndose público el código de Eclipse utilizado o modificado, pero sin la obligación de poner a disposición del público el nuevo código añadido.

Entre las características del la Licencia CPL se especifica que cualquier software bajo esta licencia puede distribuirse libremente, permitiéndose la venta sin que se requiera el pago de compensaciones de cualquier tipo. Además cualquier programa

bajo licencia CPL debe permitir la distribución en forma de código fuente y en forma compilada; si el producto no incluye el código fuente, deberá incluirse en él la manera de conseguirlo. Por último cualquier programa bajo licencia CPL debe permitir la producción de trabajos derivados a partir de él y la introducción de modificaciones en el programa original.

Con la licencia CPL, el concepto de derechos de autor sigue vigente. El copyright de los programas, el código, etc. pertenece a sus legítimos autores pero cuando un programa se distribuye bajo la licencia CPL, el poseedor de sus derechos de autor concede a cualquiera una licencia[22]. Por este motivo, no es extraño ver el copyright de IBM en la documentación de Eclipse y en el propio producto, pues desarrolló inicialmente la mayor parte de éste.

2.2.1.3. Características

Según IBM, Eclipse se define como: "una plataforma universal para integrar herramientas de desarrollo con una arquitectura abierta, basada en plug-ins". Plataforma universal, pues emplea una estructura abierta de plug-ins (extensiones) que permite expandir las capacidades de la plataforma base[23].

Eclipse se encuentra disponible para los sistemas operativos Solaris 8, HP-UX, AIX, Windows 98/ME/2000/XP, Linux SuSE 7.1, Linux Red Hat 9 y QNX.

---

[22] Open Source Initiative. *Common Public License Version 1.0*. Disponible en: <http://www.opensource.org/licenses/cpl1.0.php>

[23] Abián, Miguel Ángel. 2003. *El Archipiélago Eclipse*. Disponible en: <http://www.javahispano.org/articles.article.action?id=75>

Lo primero que se necesita hacer antes de instalar Eclipse es asegurarse de que tiene disponible un Java Runtime Environment (JRE) apropiado. Eclipse es compatible con las versiones 1.3, 1.4 y la última 1.5 del Java Development Kit (JDK). Una vez que se tenga instalado el JRE, se debe obtener el paquete de instalación del producto, desempaquetarlo y ejecutar la aplicación, no es necesario hacer configuraciones adicionales. En este proceso de instalación se completar algunas pocas tareas como la creación del directorio workspace para almacenar los ficheros de proyectos[24].

La plataforma Eclipse ofrece muchas de las características que cabría esperar de un IDE de calidad comercial: editor con sintaxis coloreada, compilación incremental, un depurador que tiene en cuenta los threads (hilos), un navegador de clases, un controlador de ficheros/proyectos, e interfaces para control estándar de código fuente, como CVS y ClearCase. Eclipse también incluye varias características únicas como la refactorización de código, o la actualización e instalación automática de código.

Eclipse se diferencia de los IDEs tradicionales en varias cosas fundamentales. Quizás lo más interesante de Eclipse es que es completamente neutral a la plataforma y al lenguaje. Además de la mezcla tan amplia de lenguajes soportados por el Eclipse Consortium (Java, C/C++, Cobol), también hay otros proyectos adicionales para añadir el soporte de lenguajes tan diversos como Python, Eiffel, PHP, Ruby, y C#.

[24] Eclipse Project. 2004. *Eclipse 3.0.1 Workbench User Guide*

Para el desarrollador de plug-ins, el IDE le proporciona un conjunto de frameworks, un conjunto de reglas de integración con la plataforma, una interface gráfica, soporte para el control de versiones, infraestructura para la depuración independiente del lenguaje usado y el control de las bibliotecas gráficas, entre otras muchas características. Los desarrolladores de plug-ins e IDEs pueden usar todas estas funcionalidades ya incorporadas para desarrollar sus propias herramientas que expandan la plataforma.

Eclipse viene incluido inicialmente por JDT (Java Development Tooling), un plug-in que extienden la plataforma al proporcionar características para la edición, compilación, depuración y ejecución de código Java. El JDT viene incluido en el SDK de Eclipse.

Eclipse posee un editor muy visual con sintaxis coloreada, ofrece compilación incremental de código, un depurador que permite establecer puntos de interrupción, modificar e inspeccionar valores de variables, un navegador de clases, un gestor de archivos y proyectos. La versión estándar de Eclipse proporciona también una biblioteca de refactorización.

También incluye una herramienta para completar código: el asistente de contenido, encargado de mostrar los métodos y atributos de las clases con las que se está trabajando sean estos APIs de Java o de cualquier otra clase dentro del proyecto, inclusive ficheros JAR. Este asistente también proporciona información de cada uno de los métodos mediante una ventana secundaria que informa de los errores

cometidos al escribir el código. Incluye también asistentes para la creación de clases e interfaces, y proporciona integración con el sistema CVS (Concurrent Version System), que sirve para llevar el control de las versiones.

Una característica poco mencionada pero muy útil de Eclipse es el soporte que ofrece a HotSwap (cambio en caliente), una de las novedades del JDK 1.4. Esta propiedad permite sustituir código o modificarlo mientras se está ejecutando un programa; esto evita parar la ejecución, realizar las modificaciones, grabar, volver a compilar y ejecutar de nuevo. Se puede también establecer puntos de interrupción en el código, ejecutar la aplicación, comprobar el estado de las variables en los puntos de interrupción cuando éstos se alcancen, modificar el código o introducir código nuevo, grabar los cambios y continuar la ejecución del programa. Los errores pueden localizarse y analizarse al vuelo, sin verse obligado a salir de la aplicación, cambiar el código, recompilar y comenzar una nueva sesión de depuración.

Por ejemplo si se trabaja con servlets y JSPs (Java Server Pages), y junto con Eclipse se utiliza algún servidor web. Debido al ciclo de vida de los servlets, se necesita parar y volver a arrancar el servidor web cada vez que se desea actualizar la clase servlet o recargar el código Java llamado por algún fichero JSP. Cuando se cambia en Eclipse un método de un servlet o de un fichero JSP, Eclipse compila incrementalmente solamente el método modificado en la clase, no la clase completa, y la liga en caliente al programa en ejecución.

Otra característica de Eclipse, muy eficiente para reducir los tiempos de depuración y pruebas, es la compilación incremental automática del código. Eclipse no cuenta con un menú de compilación pues no es necesario: cada vez que se hacen cambios en uno o más ficheros, el compilador interno de Eclipse recompila todos los ficheros fuente afectados por los cambios. En consecuencia, tampoco es necesario esperar a la compilación para detectar ciertos errores, ya que Eclipse muestra indicaciones de los errores aparecidos según se van realizando o guardando los cambios.

2.2.1.4. Interface Gráfica

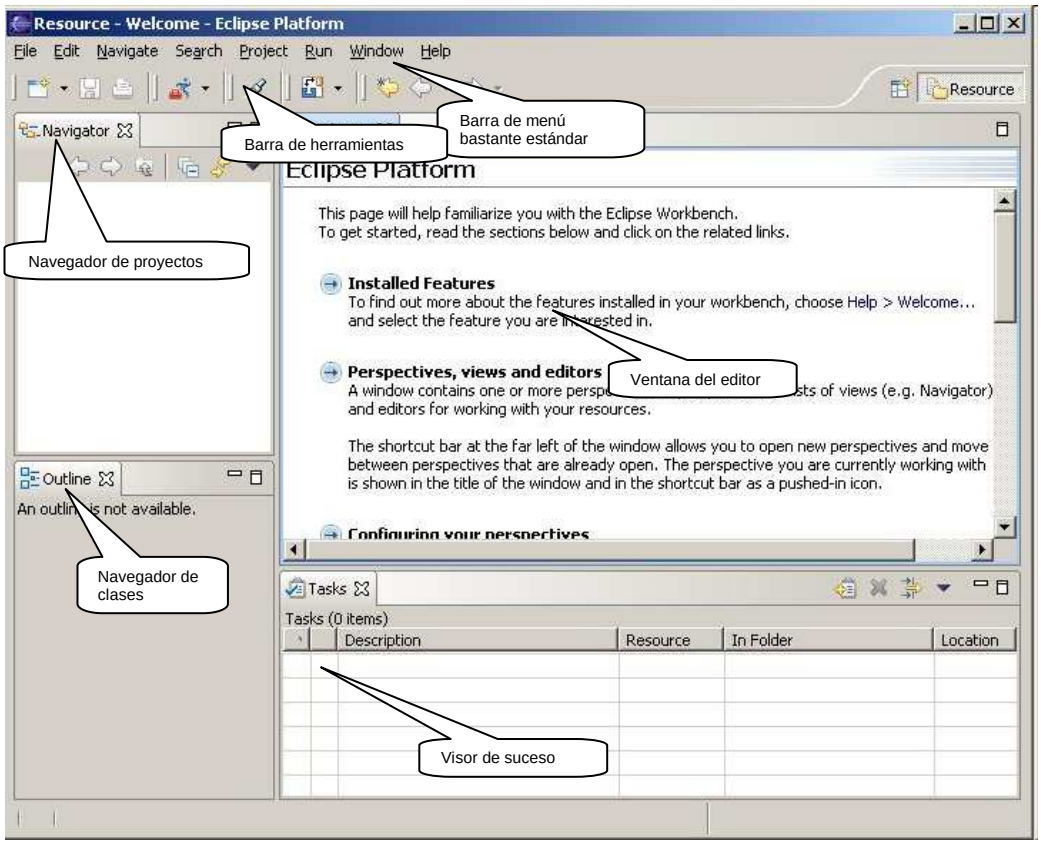


Figura 2.1: Interface Gráfica Eclipse
Fuente: Programa Eclipse
Autor: Edison Hernández

Creación de nuevos proyectos:

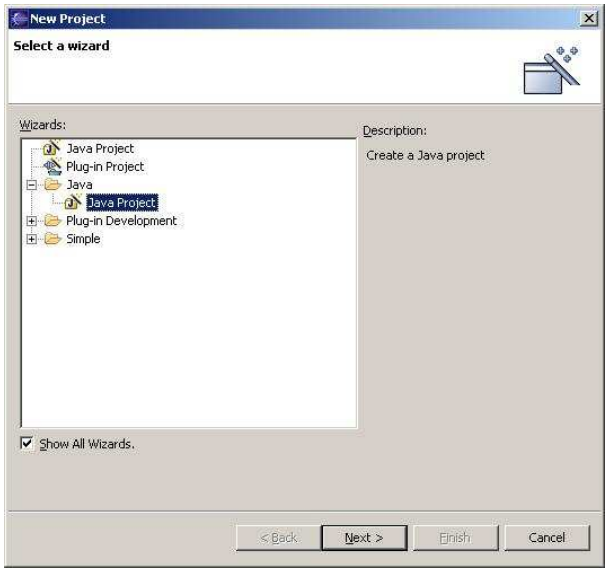


Figura 2.2: Creación de proyectos en Eclipse.
Fuente: Programa Eclipse

El *Navegador de Ficheros*, la *Lista de Tareas* y la ventana de *Code Outline* están presentes siempre en la interface. Para crear un nuevo proyecto simplemente se selecciona File->New->Project. Dependiendo de los plug-ins instalados en Eclipse, aparecen las opciones para creación de archivos.

Normalmente al crear un proyecto aparece una ventana paso a paso que ayuda durante el proceso. Una vez creado el proyecto, se puede abrir las diferentes "perspectivas Java" para visualizar los archivos de acuerdo a las preferencias del usuario. Una perspectiva (en Eclipse) no es más que distribución de ventanas. Eclipse viene con varias perspectivas por defecto (Resource, Java, Debug) que se pueden personalizar, o se pueden crear perspectivas completamente nuevas.

2.2.2. IntelliJ's IDEA

2.2.2.1. Historia

JetBrains es la compañía de desarrollo creadora de IntelliJ. Según sus palabras esta compañía se ha centrado en el desarrollo de "herramientas inteligentes".

JetBrains fue fundada en febrero de 2000 en Praga, Republica Checa. Pocos meses después fue lanzada una herramienta llamada *IntelliJ CodeSearch*. La primera versión de IntelliJ IDEA fue lanzada en enero de 2001 y su segunda versión fue lanzada en julio del mismo año.

Las versiones de la herramienta han ido progresando constantemente debido al interés y la gran acogida de la comunidad de desarrolladores, especialmente de Java, hacia la herramienta[25].

2.2.2.2. Tipo de Licencia

Este es un programa comercia por lo que tiene costo, pero permite bajarse una versión de evaluación la cual puede ser utilizada por 60 días. A pesar de ser un programa comercial, IntelliJ permite agregar extensiones adicionales para mejorar sus características internas. Estas extensiones pueden ser gratuitas o comerciales.

2.2.2.3. Características

El soporte para desarrollo de aplicaciones bajo la plataforma J2EE es bastante bueno. Gracias a esto IntelliJ permite la creación de componentes como clases Java, EJB o Java Web Services.

---

[25] JetBrains. 2004. *IntelliJ IDEA 4.5 Overview*

Al igual que Eclipse, IntelliJ ofrece varias perspectivas exclusivas para el desarrollo de aplicaciones. Otra característica es el soporte XML el cual ofrece igualmente características de auto-completar código o detección de errores de sintaxis. Posee también un editor de JSPs con características similares a las anteriores. Por último su interface a colores permite una identificación más fácil de palabras claves, variables, etc.

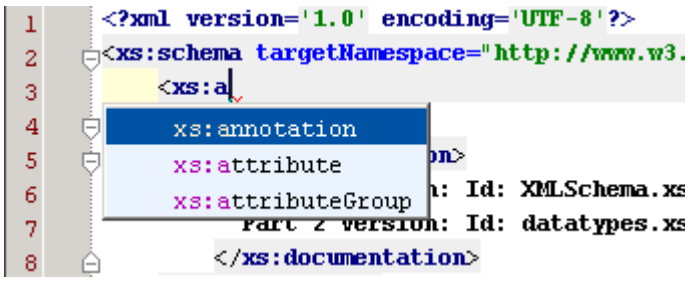


Figura 2.3: Editor de código IntelliJ
Fuente: Programa IntelliJ

Mediante IntelliJ se puede manejar la documentación de la aplicación a través de JavaDoc que es parte del JDK y que genera la documentación de manera automáticamente a través de los comentarios en el código de la aplicación. Para esto simplemente hay que definir unos pocos parámetros adicionales. Además utiliza la misma documentación de Java para incluir un inspector de código que guía al usuario a través del un archivo indicando cual es la función y característica.

Una de las ventajas del uso de esta herramienta es que cuenta con un graficador de interfaces (GUI Designer) el cual es muy útil a la hora de crear la capa de presentación ya sean para Applets Java o para aplicaciones cliente/servidor.

Se integra con CVS para el control de versiones de los proyectos. Una característica importante es que cuenta con un historial local de la aplicación, lo que permite hacer un seguimiento de los cambios realizados en el código de un archivo del proyecto, así mismo como cambios hechos en el contenido o en la estructura de directorios. Esta característica previene al desarrollador de hacer modificaciones o cambios accidentales, incluso si los cambios son hechos fuera del IntelliJ IDEA.

IntelliJ permite hacer compilaciones de código mediante su interface. Es posible especificar varios JDK (Java Development Kit) si es que se cuenta con diferentes versiones. Al momento de la compilación, IntelliJ muestras una lista bastante detallada de los errores existentes en un archivo y en el proyecto en general.

Las últimas versiones de IntelliJ soportan Java SDK 1.5 pero trabaja igualmente bien para versiones anteriores de esta plataforma de desarrollo.

IntelliJ provee de un conjunto de APIs que permiten la creación de extensiones para mejorar las prestaciones de la herramienta. Además provee un conjunto de API para integrar sistemas de control de versiones CVS y servidores de aplicaciones.

2.2.2.4. Interface Gráfica

La interface de la aplicación presenta varios paneles para manejo de las diferentes etapas y funciones las cuales pueden ser escondidas y mostradas a gusto. Además se puede cambiar entre paneles sin necesidad de cerrarlos. Muchas de las características

de la aplicación pueden ser manejadas únicamente con el teclado lo que es muy útil. Estas funciones son configurables en el panel de preferencias de la herramienta.

IntelliJ permite manejar varios proyectos a la vez y permite crear ventanas flotantes para que se pueda trabajar en varios objetos al mismo tiempo. Todos los componentes de la interface pueden ser designados como ventanas flotantes; por ejemplo los navegadores del proyecto, buscadores, consolas de eventos, inspectores de resultados, copiladores o los archivos de cada proyecto.

El editor de código soporta varios tipos de codificación como Java, JSP, XML o HTML. Cada uno de estos editores ayudan al programador con características como auto-completar o detección de errores de sintaxis en el código. Todas estas características son configurables de acuerdo con las necesidades y preferencias del usuario.

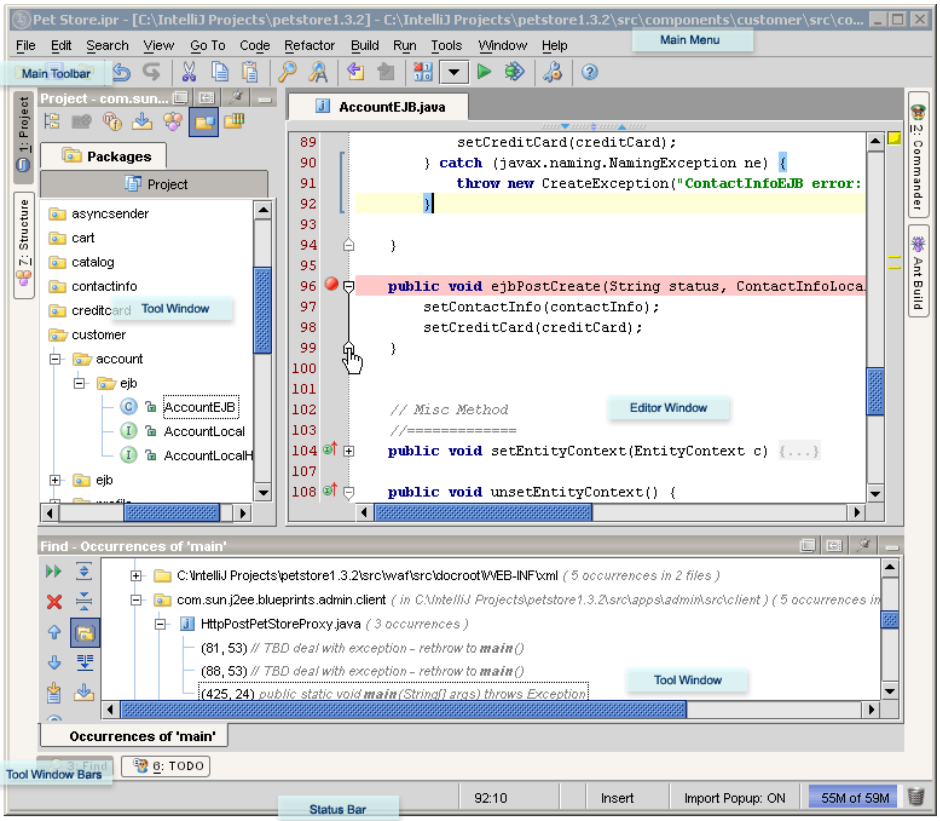


Figura 2.4: Interface de IntelliJ's Idea
Fuente: Programa IntelliJ
Autor: Edison Hernández

IntelliJ's IDEA cuenta con un depurador de código bastante útil. Además soporta *HotSpot* el cual permite hacer cambios en el código en etapas de ejecución y depuración. Permite definir *breakpoints* (puntos de quiebre) para hacer un seguimiento paso a paso en etapas de ejecución y así optimizar el código. Estos *breakpoints* pueden ser de tres tipos: *Method entry/exit breakpoints* permiten hacer seguimientos de métodos, entradas y eventos, *Exception breakpoints* permiten configurar acciones que manejan excepciones de código y errores, y por último *Conditional breakpoints* permiten especificar *breakpoints* condicionales si se dan condiciones dadas.

2.2.3. Sun Java Studio

2.2.3.1. Historia

Sun One Studio ha ido evolucionando constantemente a través de los años. Creada por Sun Microsystems, Sun One Studio es considerada una de las herramientas más completas para el desarrollo de aplicaciones Java.

Primero Sun Microsystems lanzó una IDE para Java llamado *Forte for Java* el cual era una de las herramientas más usadas de su época, aunque por ser construida sobre un núcleo Java, tenia muchas falencias con respecto al rendimiento sobre sistemas operativos Windows.

Posteriormente Forte for Java evolucionó hacia Sun One Studio. Una herramienta integrada que permite construir aplicaciones utilizando la plataforma de J2EE. Al tener J2EE y Sun One Studio el mismo creador, se considerado que trabajan muy bien juntas y Sun One Studio intenta sacar provecho de todas las prestaciones que ofrece la plataforma J2EE[26].

La última versión de Sun One Studio fue llamada Sun Java Studio cuya versión final es la *5 update 1*.

2.2.3.2. Tipo de Licencia

Esta es una herramienta comercial, aunque existe una versión de evaluación la cual puede ser bajada de la página web de Sun Microsystems. Esta herramienta es

[26] Sun Microsystems. *Sun™ ONE Studio 5 Standard Edition Getting Started Guide*

considerada “cerrada” ya que no permite hacer cambios al la funcionalidad, ni agregar extensiones adicionales de otros fabricantes.

2.2.3.3. Características

Sun Java Studio es un IDE considerado una herramienta muy poderosa para el desarrollo de aplicaciones Java, con un conjunto de características que ayudan en las etapas de creación de una aplicación, es utilizado especialmente con la plataforma J2EE en integración con Web Services.

Esta herramienta esta encaminada hacia la creación de aplicaciones empresariales utilizando la arquitectura NetBeans de Sun Microsystems la cual permite agregar algunos plug-ins (creados o aprobados por Sun Microsystems) a la aplicación. Aunque está diseñado para trabajar con la plataforma J2EE, también permite construir aplicaciones más simples utilizando la plataforma J2SE. Viene con un servidor de aplicaciones llamado Sun Java System Application Server (anteriormente llamado Sun ONE Application Server) el cual es utilizado para ejecutar aplicaciones J2EE y Web Services. Esta herramienta además permite crear aplicaciones para dispositivos móviles utilizando la plataforma de Java J2ME mediante un componente adicional llamado Sun ONE Studio, Mobile Edition.

Esta herramienta corre sobre varios sistemas operativos, en su última versión incluyen Solaris 8 o 9 (SPARC Edition), Solaris 9 (x86 Platform edition), Red Hat Linux 7.2 y Windows en sus versiones NT, 2000 o XP.

Sun Java Studio es considerada una herramienta que simplifica las etapas de pruebas y producción de una aplicación así como la etapa de programación, ya que posee funciones que ayudan en la creación Web Services de manera más fácil. A pesar de la variedad de opciones que ofrece Sun Java Studio, no soporta modelado UML, refactorización o soporte para Struts.

Presenta platillas para la creación de varios tipos de clases (EJBs, clases Java, controladores, etc), además de páginas JSP o Web Services, dichas plantillas trabajan como modelos base para la creación de los diferentes objetos, simplificando su creación y codificación. El editor de código que posee la herramienta presenta menús contextuales que permiten la creación de constructores, métodos y clases Java.

A través de la interface gráfica se puede generar código para la construcción de interfaces AWT o Swing pero únicamente se puede trabajar con una de ellas a la vez. Aunque las versiones anteriores de Sun One Studio no soportaban la creación de interfaces web la última versión ya posee esta opción. Además presenta componentes que ayudan en la etapa de pruebas. Con respecto al código, este IDE ofrece la posibilidad de auto-completar código y permite la creación de documentación a través de JavaDoc. En sus últimas versiones a implementado el manejo de CVS.

El rendimiento de la aplicación es un factor importante a la hora de elegir una herramienta de desarrollo y es aquí donde Java Studio flaquea. El rendimiento es generalmente bueno, pero especialmente sobre Linux la velocidad decae drásticamente. Indistintamente de la versión, el rendimiento en los sistemas

operativos basados en UNIX es bajo, considerando que Sun es la casa desarrolladora de UNIX. En Windows es donde se tiene mejores prestaciones de rendimiento[27].

2.2.3.4. Interface Gráfica

Para la creación de interfaces gráficas de usuario (GUI) Sun Java Studio viene con una extensión llamada Sun Java Studio Creator. La interface de usuario de esta herramienta es muy intuitiva y es por esto que la última versión de Sun Java Studio tiene gran aceptación.

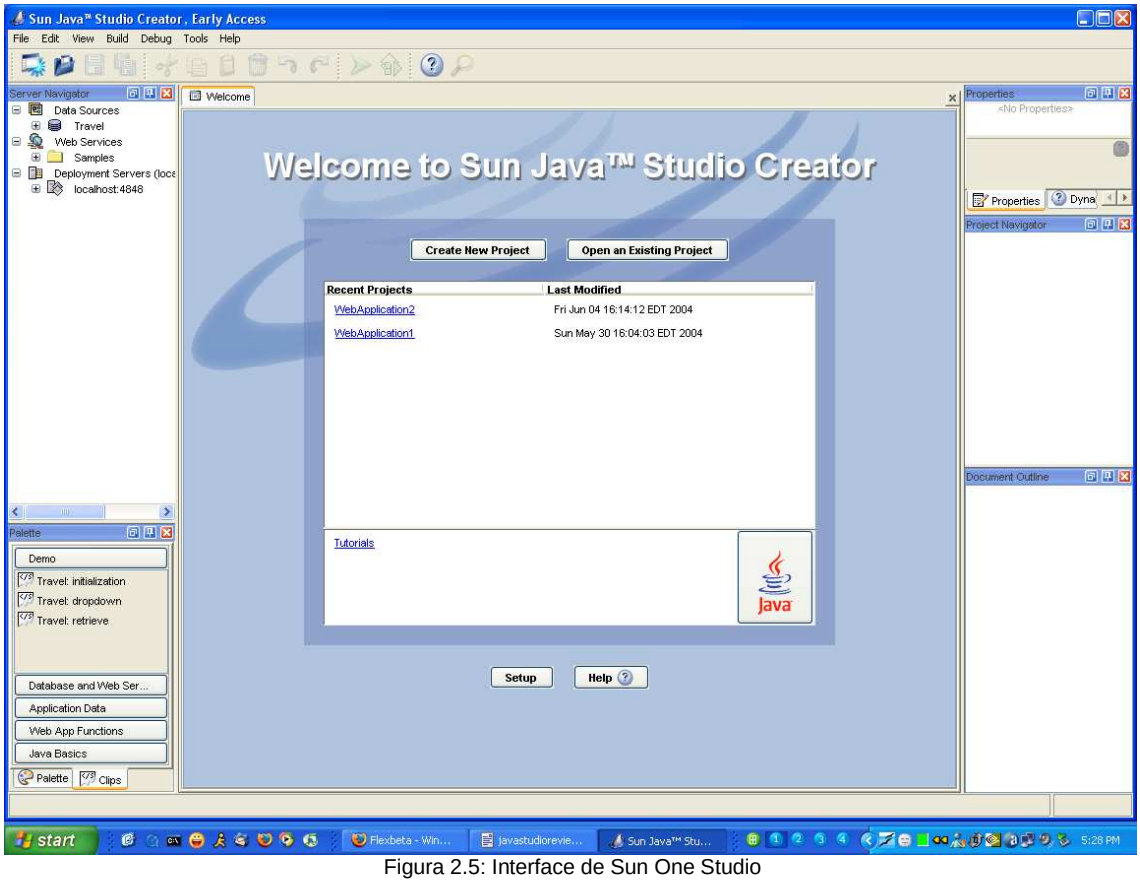


Figura 2.5: Interface de Sun One Studio
Fuente: Programa Sun One Studio
Autor: Edison Hernández

[27] Stephens, Matt. *First Look: Sun Java Studio*. Disponible en: <http://www.softwarereality.com/reviews/creator_ea.jsp>

Se puede decir que el diseño es muy parecido a la de Visual Studio, ya que presenta los componentes en sección de capas (layouts) y una barra de herramientas con los objetos más comúnmente utilizados divididos secciones (User Defined, JSF Compnentes y JSF Validator). La interface es similar entre las diferentes plataformas (Windows, Linux o Solaris). Permite interactuar entre el diseño y el código de una aplicación sin problemas.

La barra de herramientas en general es bastante grande y colorida, esto puede gustar al inicio pero después puede crear un ambiente inadecuado para la programación. A pesar de esto existen grandes mejoras en la interface de Java Studio con respecto a sus versiones anteriores (por ejemplo en los exploradores de archivos y proyectos).

Presenta varios tipos de navegadores de archivos entre los que destaca la llamada *vista lógica* que muestra los archivos de un proyecto de acuerdo a su categoría en lugar de su ubicación. Estas categorías son Web Pages, Java Sources, Resources (imágenes, hojas de estilo, etc) y referencias (librerías o clases externas al proyecto). Esto es muy útil y no muchas herramientas ofrecen esta opción.

Con Java Studio no es posible compilar un archivo individual, es necesario compilar todo el proyecto, afortunadamente esta compilación es incremental, lo que quiere decir que solo las clases que han sido cambiadas y necesitan compilación son recompiladas.

El editor de páginas web es bastante bueno. Permite agregar componentes gráficos mediante drag-and-drop (arrastra y soltar) al más puro estilo del mejor editor web que existe (Macromedia Dreamweaver). Mediante este editor se puede manejar igualmente las hojas de estilo o las plantillas de diseño.

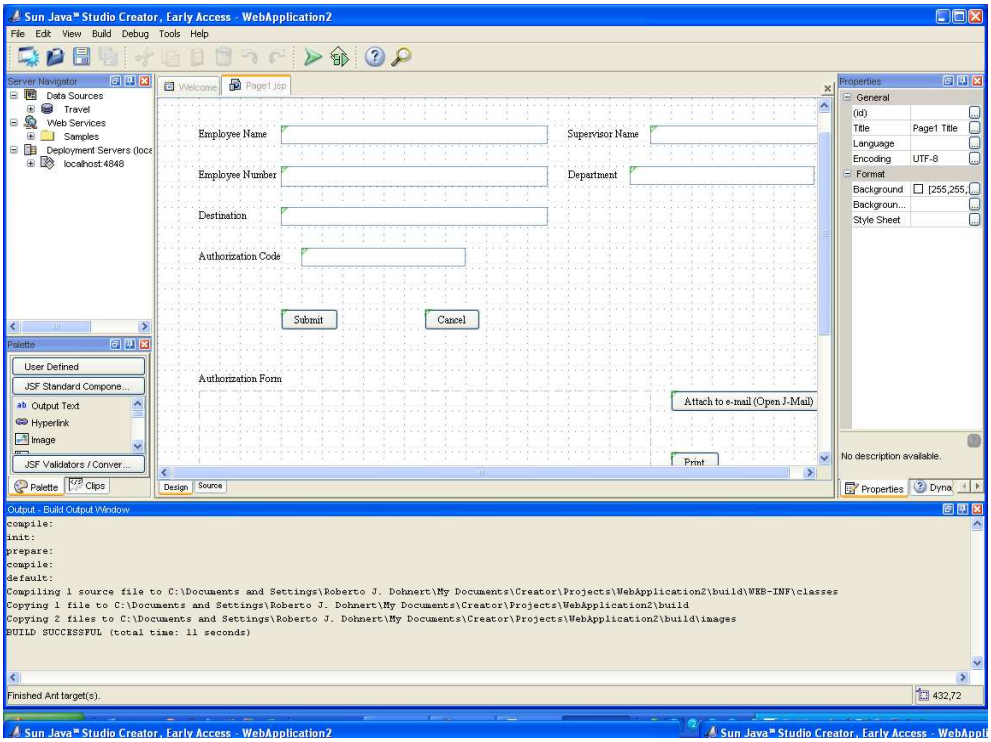


Figura 2.6: Editor de páginas web en Sun One Studio
Fuente: Programa Sun One Studio
Autor: Edison Hernández

En la parte inferior del editor existen tabuladores que permiten cambiar entre las vistas de código y diseño pero no permite tener una vista compartida. La vista de diseño se actualiza automáticamente al momento de algún cambio en el código. Es bastante completa pero presenta varias fallas, por ejemplo si se corre un componente en la vista de diseño, se cambia a la vista de código y se vuelve a la vista de diseño, el componente vuelve a aparecer mágicamente.

2.2.4. IBM WebSphere

2.2.4.1. Historia

La actual generación de herramientas de desarrollo IBM consiste en WebSphere Studio Application Developer, WebSphere Studio Site Developer, WebSphere Studio Enterprise Developer y Websphere Studio Workbench y todas se basan en Java. IBM está reemplazando, tanto interna como externamente, sus productos de la familia Visual Age y WebSphere Studio por los nuevos productos basados en Eclipse. Al momento ya no se producen nuevas versiones de Visual Age for Java aunque se continúa dando soporte.

Los motivos de esta nueva estrategia basada en Eclipse obedecen a una suerte de relevo generacional tecnológico: Visual Age for Java y Websphere Studio se crearon en una época en la que las páginas web eran páginas HTML servidas por CGIs (Common Gateways Interfaces). En aquel entonces, gracias a los applets, Java alcanzó una gran popularidad en el lado del cliente. El panorama actual apenas guarda parecido con el de hace diez años: las páginas web actuales incluyen vídeo de alta resolución, animaciones, sonidos, HTML dinámico. También para Java ha pasado el tiempo; los applets han caído en desuso, y el lenguaje se ha hecho fuerte en el lado del servidor. La plataforma J2EE, que comenzó a existir cuando ya se había lanzado las primeras versiones de Visual Age y WebSphere Studio, ha crecido mucho y en direcciones muy distintas como servlets, JSPs, EJBs o XML

2.2.4.2. Tipo de Licencia

Esta es una herramienta comercial la cual debe ser adquirida pagando un costo para poder ser utilizada. Lo notable es que permite agregar extensiones adicionales con gran libertad.

2.2.4.3. Características

IBM ofrece un conjunto de herramientas que compiten con los diferentes IDEs del mercado. La solución que ofrece IBM es una familia de herramientas para desarrollo de aplicaciones llamada WebSphere Workbench, que sirve para manejo de servicios J2EE, servidores Web, y servidores de base de datos. WebSphere Studio funcionan sobre una base de herramientas *Open Source* descendientes de la plataforma Eclipse. WebSphere está disponible para múltiples sistemas operativos, se traduce a muchos idiomas para la distribución mundial, y es accesible para personas con dificultades físicas. WebSphere Studio Site Developer y WebSphere Studio Application Developer han sido los primeros productos de la nueva familia WebSphere Studio en ser lanzados.

WebSphere Studio Application Developer va varios pasos por delante del Site Developer, ya que permite el desarrollo de aplicaciones J2EE y de bases de datos en un entorno de programación conjunto. Incluye todas las funciones del Site Developer y permite la creación de EJBs, incorpora asistentes para conexiones a bases de datos y permite usar el servidor de aplicaciones propio de IBM llamado WebSphere Application Server. Entre sus capacidades más notables está el asistente para XML y

para manejo de bases de datos. WebSphere Application Developer es una de las herramientas más completas que existen en el mercado[28].

WebSphere Application Developer es compatible con el estándar de Eclipse con respecto a sus plug-ins; sean estos propietarios, open source o free software, por lo que aplicaciones hechas en WebSphere Application Developer pueden adaptarse a Eclipse. Los productos comerciales de IBM son derivaciones concretas de Eclipse.

IBM considera que hay 2 métodos de desarrollo de aplicaciones enfocadas en el middleware (capa para conectar aplicaciones que puedan correr en diferentes plataformas o provenientes de diferentes vendedores): el método direct-to-middleware y el método model-driven. En el primero, el desarrollo de la capa intermedia se la deja a cargo del desarrollador de la aplicación. El segundo método utiliza herramientas que operan exclusivamente sobre los formatos de archivos de componentes definidos por el modelo de programación de la capa intermedia. Esto da a la aplicación acceso total a las características del modelo de programación de la capa intermedia.

Herramientas direct-to-middleware correctamente diseñadas pueden simplificar la programación y reducir la carga impuesta a los desarrolladores. Por definición WebSphere Studio utiliza el método direct-to-middleware[29].

---

[28] Budinsky, F; DeCandio G. 2004. *WebSphere Studio Overview*
[29] Sun Microsystems. *Sun Java System Studio Standard 5 update 1 Tutorial*

WebSphere Studio viene con extensiones para manejo de las diferentes capas de desarrollo (por ejemplo la capa da usuarios). WebSphere Studio Application Developer Integration Edition extiende Application Developer con instrumentos para el desarrollo de aplicaciones de integración de negocios. WebSphere Studio Enterprise Developer ofrece instrumentos para desarrollo de ambientes del mainframe IBM. WebSphere Development Studio Client maneja IBM eServer* iSeries* para integración de componentes iSeries y aplicaciones distribuidas.

2.2.4.4. Interface Gráfica

WebSphere Studio provee una interface de usuario que es considerada muy fácil de utilizar y que permite diferentes combinaciones y escenarios para desarrollo. WebSphere Studio utiliza una metodología de desarrollo llamada User-Centered Design. Esta metodología posiciona al usuario en el centro del ambiente de desarrollo y le permite intervenir en todas las etapas y capas de la aplicación, todo esto mediante el manejo de extensiones.

Los menús y barras de herramientas son bastante parecidos a los que presenta la interface gráfica de Eclipse y también mediante extensiones adicionales se puede agregar nuevas vistas, editores, menús y botones.

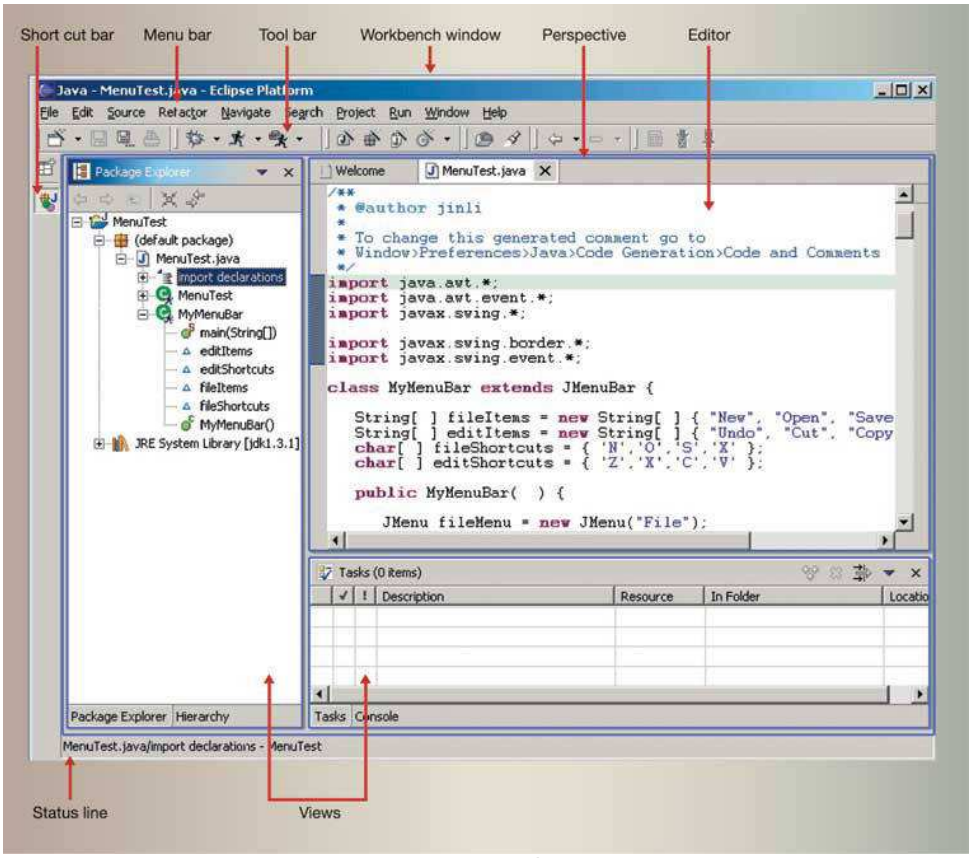


Figura 2.7: Interface de la aplicación WebSphere Studio
Fuente: Manual Oficial WebSphere
Autor: IBM

Cada escenario de desarrollo provee de menús y barras estándares, dependiendo del tipo de proyecto que esta trabajando. El usuario está en la capacidad de mostrar o esconder cada una de estos componentes de acuerdo a su preferencia. Adicionalmente ofrece varias perspectivas de desarrollo que se muestran de acuerdo al tipo de componente en el que se está trabajando.

La interface de WebSphere provee componentes como *Quick Fixes* o *Quick Assist* que ayudan al usuario a identificar problemas en etapa de programación, y sugiere alternativas para solucionar estos problemas. WebSphere Studio además presenta diferentes tipos de vistas de archivos entre las que se destacan la vista lógica y de herencia que permiten identificar y ubicar archivos de manera más rápida.

El escenario de desarrollo que ofrece WebSphere Studio le permite al usuario participar adicionalmente en las siguientes etapas de desarrollo:

- Implementación de funciones para reportes de bases de datos y publicación como Web Services. Esta fase utiliza *WebSphere Studio SQL query* para manejo de procedimientos almacenados y datos de la base.
- Permite definir Web Services para llevar a cabo actualizaciones de registros en bases de datos. Estos servicios pueden interactuar con EJB's.
- Permite crear EJB's mediante la utilización de WebSphere Studio's UML y editores java existentes. Son utilizados especialmente para Session Beans.

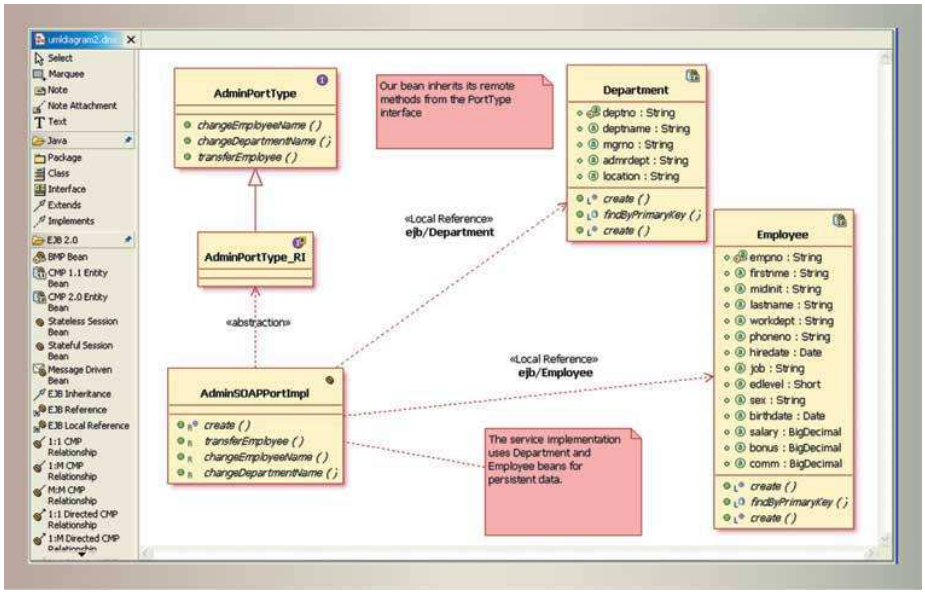


Figura 2.8: Soporte UML en WebSphere
Fuente: Manual Oficial WebSphere
Autor: IBM

- Pruebas y depuración de aplicaciones. Esta fase explota las características de los servidores utilizados para la creación de aplicaciones web.
- Creación de aplicaciones Web. Ofrece una interface de usuario fácil para el desarrollo de todos los componentes de una aplicación Web (JSPs o Servlets).

- Análisis y mejora de desempeño de la aplicación. Ofrece herramientas como *Performance Profiling* para identificar cuellos de botella y problemas.

Una de las características principales es el ambiente de diseño de páginas web. Incluye un editor web que permite la creación, edición y mantenimiento de páginas web. Este editor llamado *Page Designer* utiliza el método *drag-and-drop* para agregar los diferentes componentes que posee una página web. Hay que tomar en cuenta que este editor está dirigido hacia el desarrollo de páginas JSP exclusivamente por lo que no permite el desarrollo de páginas basadas en otros servidores de aplicaciones (PHP o ASP). Por último permite el manejo de Templates (plantillas de diseño) que permiten crear un diseño base para todas las páginas de un sitio web.

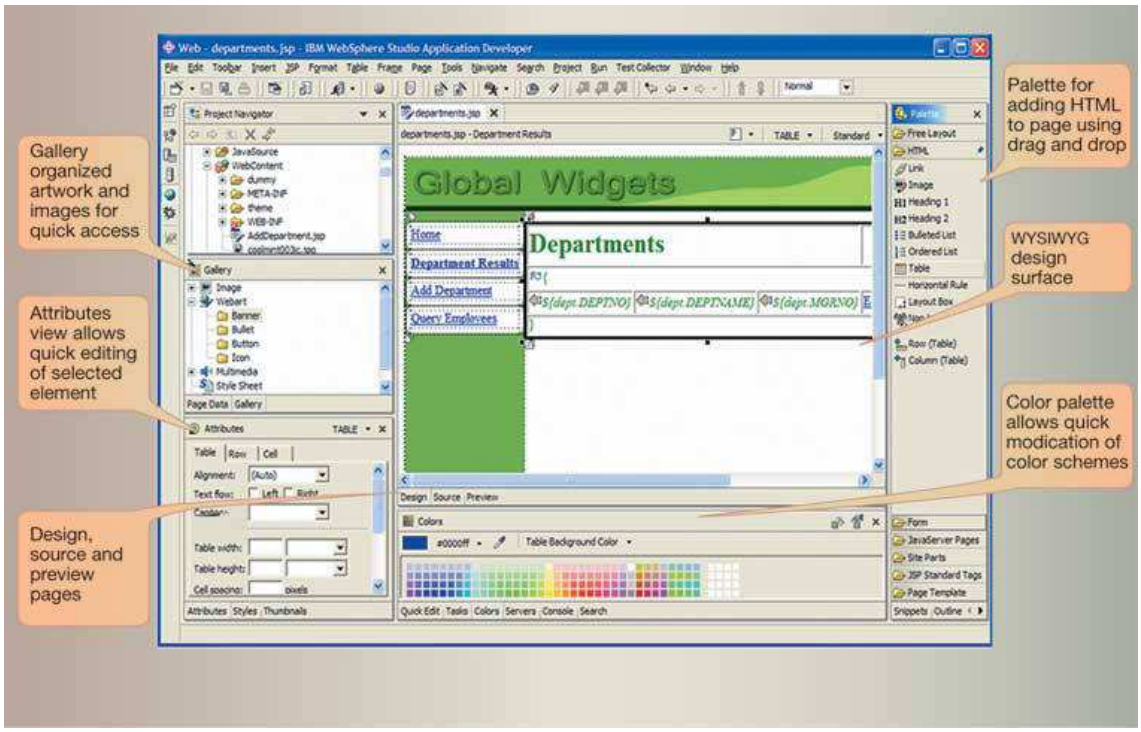


Figura 2.9: Generador de Interfaces Web en WebSphere
Fuente: Manual Oficial WebSphere

2.3. VENTAJAS Y DESVENTAJAS

A continuación un cuadro que muestra las principales ventajes y desventajas de las cuatro herramientas detalladas anteriormente:

Herramientas	Ventajas	Desventajas
Eclipse	La principal ventaja es lo gratuito de la herramienta. Tiene licencia libre por lo que es posible hacer modificaciones en el código. Es relativamente liviano, no ocupa muchos recursos del sistema a pesar de que su desarrollo se hizo utilizando Java. La inclusión de plug-ins es otra poderosa ventaja sobre otras herramientas, ya que permite agregar componentes cuando se desee. Los plug-ins también pueden ser creados libremente si existe una necesidad específica. Eclipse puede trabajar con varios códigos de programación como CGI, PHP, etc., mediante extensiones adicionales.	Eclipse no incluye un diseñador visual de interfaces gráficas (GUI builder). Además Eclipse no incorpora plug-ins nativos para comunicarse con servidores de aplicaciones. Falta soporte para Enterprise JavaBeans, Java Server Pages y Servlets; y no se incluyen asistentes para la edición o depuración de estos. La incorporación de plug-ins puede considerarse tanto una ventaja como una desventaja, ya que su inclusión puede provocar conflictos con otros componentes.
IntelliJ's IDEA	Es considerada una herramienta con los mejores asistentes de código, ya que incrementan la velocidad y facilita la programación. Posee una buena interface gráfica. Es una herramienta muy buena para la etapa de depuración.	Es totalmente orientada al proyecto. Tiene un precio, que aunque no es alto, se puede considerar una desventaja.
Sun Java Studio	Viene con varios componentes extras al momento de instalar la herramienta. Aunque muchos de estos pueden ser inútiles para la mayoría de usuarios, sus presentaciones y ventajas en el desarrollo de aplicaciones corporativas pueden facilitar la vida del programador.	El precio es alto aunque la herramienta es muy completa. La herramienta puede ser muy pesada para una computadora estándar, ya que necesita grandes requerimientos de memoria y procesador para funcionar correctamente. Esto puede ser frustrante para el programador especialmente en etapas de compilación y depuración.
IBM WebSphere	Es considerada una herramienta bastante estándar, y bastante usada también. Esta herramienta puede trabajar conjuntamente con otras herramientas de Java como por ejemplo NetBeans. Lo principal es que ofrece un editor JSP/HTML bastante bueno, que permite insertar componentes visuales. Permite tener vistas previas de las páginas web.	WebSphere Studio Application Developer proporciona un diseñador gráfico bastante más simple, y es muy difícil hacer complejas interfaces gráficas con esta herramienta. Su precio es alto.

Cuadro 2.1: Ventajas y desventajas de diferentes IDEs
Autor: Edison Hernández

2.4. ANÁLISIS ECONÓMICO

Si duda la mejor opción cuando se refiere al aspecto económico es Eclipse, ya que es una herramienta *Open Source Freeware* a la que es posible agregar extensiones adicionales. Si se esta contento con el rendimiento se puede realizar una donación económica voluntaria. Además si en el equipo de desarrollo van a trabajar varias personas, no hay que preocuparse por el costo de cada una de las licencias que se deben adquirir

IntelliJ IDEA tiene un costo en su última versión (4.5.2) de $499 por. Aunque el costo es más bajo que en otras herramientas, al haber opciones freeware, se debe pensar si se quiere pagar tanto por una herramienta, especialmente si se necesitan más de una licencia.

Sun Java Studio cuesta $695 por licencia Standard, $495 por actualización pero hay que tomar en cuenta que estos precios son de la versión estándar. Existe otra versión mucha más avanzada llamada Sun Java Studio Enterprise que tiene un costo más alto debido a sus características adicionales como por ejemplo un servidor web integrado.

IBM WebSphere Studio posee diferentes partes, si nos enfocamos al desarrollo de aplicaciones esta viene en 2 versiones, la versión para desarrollo de aplicaciones Cliente/Servidor (IBM WebSphere Studio Site Developer) tiene un costo de $740, la versión para desarrollo de aplicaciones web (IBM WebSphere Studio Application Developer) tiene un costo de $650.

Grandes empresas y corporaciones no se preocupan mucho por los costos de las herramientas para desarrollo que posee, y para estas empresas es mucho más importante el soporte que adquieres al comprar la licencia que su precio. Bajo estas circunstancias una empresa debería adquirir herramientas como las que ofrecen IBM o Sun. El servicio de soporte técnico 24/7 (24 horas, 7 días) es crucial en equipos de desarrollo de grandes empresas.

Pero existen pequeñas empresas a las cuales les resulta muy difícil adquirir licencias de productos a costos tan altos, especialmente si posee equipos de trabajos numerosos en los cuales se debe adquirir una licencia por cada persona del equipo. Esto incrementa enormemente los costos de desarrollo. En los últimos tiempos gracias al surgimiento de herramientas gratuitas muchas empresas se están encaminando hacia aplicaciones gratuitas, especialmente porque estas herramientas ofrecen iguales o mayores prestaciones que las herramientas comerciales.

Un ejemplo de esto es Eclipse, que se mantiene gracias a una comunidad de desarrolladores y aportes de IBM. Es una herramienta muy simple pero a la vez completa y la cual permite la inclusión de extensiones que la hacen mucho más potente aun. Sin duda es la mejor solución para desarrollo no solo de aplicaciones Java sino para el trabajo en diferentes plataformas, y, ya que el mundo esta en tendencias hacia el software libre, esta es una herramienta con gran futuro.

2.5. COMPARACIÓN ENTRE LAS DIFERENTES PROPUESTAS

Los desarrolladores que han trabajado con WebSphere notan algunas similitudes al trabajar con Eclipse. Muchos de los asistentes son iguales y también algunos botones. Sin embargo, internamente hay diferencias entre las que resaltan que Eclipse se basa en ficheros individuales, muchos de los cuales tienen formato ASCII, lo que hace más liviana la aplicación.

Eclipse e IntelliJ, a diferencia de otros IDEs, permiten elegir para la ejecución la máquina virtual Java que se desee, desde la versión 1.1.7 del JRE a la versión 1.5. Esta posibilidad resulta muy ventajosa para los programadores, ya que se puede hacer pruebas con el JRE que vaya a emplear el usuario final.

WebSphere permite la generación y distribución de Enterprise JavaBeans, Java Server Pages y servlets. La última versión de WebSphere incluye un entorno de pruebas (WebSphere Test Environment), un servidor EJB, un depurador integrado para servlets, y un monitor de ejecución para JSPs. Estas capacidades están ausentes en otros IDE's como Eclipse o IntelliJ.

La diferencia más importante entre Eclipse y las otras herramientas analizadas es el precio. Se tiene que pagar un alto precio por la mayoría de estas herramientas, lo que hace a Eclipse tan atractivo. Esto puede ser una gran ventaja si se tiene limitaciones de presupuesto a la hora de elegir la herramienta. A pesar de esto el costo de IntelliJ IDEA es muy accesible si se considera las prestaciones que ofrece. Aunque muchas

de las herramientas comerciales tiene un sinnúmero de características que puede considerarlas útiles, Eclipse cuenta con una amplia comunidad de desarrolladores junto a él, lo que provee de soporte y componentes adicionales para cubrir las falencias que pueda tener la herramienta.

A continuación se presenta un cuadro comparativo de las principales características de las herramientas analizadas.

	Eclipse	**IntelliJ IDEA**	**Sun Java Studio**	**IBM WebSphere**
Creador	Eclipse Project	JetBrains	Sun Microsystems	IBM
Licencia	Open Source, Licencia CPL	Licencia comercial	Licencia comercial	Licencia comercial
Costo	No tiene costo, se pueden hacer donaciones voluntarias	$499 por usuario licencia nueva $299 por actualización	$695.00 por usuario licencia nueva Standard $495.00 por actualización licencia Standard	IBM WebSphere Studio Site Developer: $740 IBM WebSphere Studio Application Developer: $650
Sistemas operativos	Windows 2000/ME/XP, Linux, Unix (Solaris), Mac OS, HP-UX,	Windows 2000/XP, Mac OS X, Linux, Unix	Windows 2000/XP, Linux, Unix Solaris	Windows NT/2000/XP, Linux.
Editor con sintaxis	SI	SI	SI	SI
Auto-ayuda	SI	SI	SI	SI
Compilación Integrada	SI	SI	SI	SI
HotSpot	SI	SI	SI	NO
Editor de GUI	NO (Extensión adicional)	SI	SI	SI
Editor de Interfaces Web integrado	NO (Extensión adicional)	NO	SI	SI
Capacidad Plug-Ins	SI	SI	NO	SI
Integración UML	SI	Si	NO	NO

Cuadro 2.2: Comparación entre diferentes IDEs
Autor: Edison Hernández

2.6. RESUMEN

Inicialmente las técnicas de programación se limitaban al uso de editores de texto simples como Notepad en Windows para escribir el código de un programa. Para poder compilar un programa se necesitaba hacerlo mediante líneas de comando, esto también sucedía para ejecutar un programa, además que no existían depuradores gráficos. La escritura de código era lenta y tediosa, la detección de errores se daba en etapas finales y además la manipulación del código era bastante difícil.

Los IDEs (Integrated Development Environment) son herramientas que no solo ayudan a programar, sino que asisten en todas las etapas de desarrollo. Por ejemplo un IDE permite editar y auto-completar el código resaltando palabras claves; además mediante la misma herramienta se puede ejecutar y depurar el programa, mostrar ayudas en línea y documentación existente, detectar errores de código mientras se escribe, manejar todos los directorios y archivos de un proyecto, o ejecutar paso a paso una aplicación.

El manejo integrado del proyecto puede ser muy útil, ya que permite manejar directorios y archivos en un solo ambiente, además de poder construir y empaquetar los diferentes componentes. También presentan control de versiones.

Ayudas al escribir código son importantes a la hora de programar. Un IDE debe identificar posibles problemas, inconsistencias o expresiones incorrectas. Además

debe poseer métodos para auto-completar código y sugerir soluciones cuando un problema se presenta.

Eclipse es una herramienta gratuita “open source” que posee similares prestaciones a las de otras herramientas de distribución comercial como Sun One Studio o WebSphere. Todas las opciones del mercado permiten la generación de código automático, compilación, depuración. La única falencia de Eclipse se podría considerar la carencia de un editor de interfaces gráficas pero esto esta por solucionarse en las próximas versiones de la herramienta. Las herramientas comerciales son muy “cerradas” con respecto a su arquitectura pero Eclipse o IntelliJ’s IDEA tienen la capacidad de poder integrarse con extensiones para mejorar su funcionalidad.

Capítulo 3

3. SALMON SOFIA

3.1. INTRODUCCIÓN

SOFIA (Salmon Open Framework for Internet Applications) es una herramienta gratuita creada por la empresa Salmon LLC para desarrollar aplicaciones en J2EE. Provee una metodología que intenta integrar todas las etapas involucradas en el desarrollo de una aplicacion web.

Salmon LLC fue fundado en el año de 1995 y se especializó en el desarrollo de aplicaciones Java haciendo el uso de la orientación a objetos, y además de herramientas y técnicas para Java de la época. Con oficinas en Nueva York, Londres y Sydney la compañía ha tratado de mantener vínculos de colaboración para la creación de soluciones y servicios de tecnología. Desde el lanzamiento de SOFIA, se han registrado más de 70000 bajadas del framework desde la página oficial de Salmon y se ha recibido gran apoyo de la comunidad de código libre. SOFIA es considerado muy popular en la comunidad de desarrolladores Java por su facilidad y rapidez[30].

SOFIA en realidad nació varios años atrás como una herramienta de desarrollo que utilizaba internamente Salmon para el desarrollo y, a decir de sus creadores, fue concebida al querer encontrar una herramienta ideal que permita el desarrollo aplicaciones web basadas en Java. Muchas de las herramientas de la epoca eran

[30] Salmon LLC. *Salmon LLC Company Approach*. Disponible en: <http://www.salmonllc.com>

pobremente creadas, con muchas fallas y errores según afirman. Según Salmon, la única posibilidad de encontrar una herramienta acorde a las necesidades de la empresa era construyéndola ellos mismos. La primera versión de SOFIA incluía componentes GUI (Graphic User Interface), programación para manejo de eventos y un componente para acceso a la base de datos. Era bastante buena pero le faltaba un editor grafico (GUI), ya que esto se lo hacía con editores de texto[31].

Inicialmente se mantuvieron alejados de JSP ya que era común mezclar código Java con HTML en servlets y otros componentes. Posteriormente una nueva versión de JSP resolvió esto al incluir las librerías de etiquetas personalizables (custom tag libraries) que se utilizaron en SOFIA para crear componentes gráficos. Después se dieron cuenta que Dreamweaver tenia la capacidad de manejar extensiones adicionales por lo que utilizaron esta herramienta para crear la interface de usuario.

En ese tiempo utilizaban IBM Visual Age Java como editor de código ya que lo consideraban el mejor de su tiempo pero posteriormente IBM abandonó ese producto por lo que tuvieron que buscar otras opciones. Para entonces Salmon había esta utilizando SOFIA por varios años sin que encontraran un IDE de su agrado, pero finalmente aparecieron 2 propuestas que les atrajo: IntelliJ's y Eclipse. Entonces combinaron este conjunto de herramientas para hacer el framework que se conoce ahora: SOFIA, Eclipse, Tomcat y Dreamweaver.

[31] Salmon LLC. *The SOFIA Philosophy*. Disponible en: <http://www.salmonllc.com>

3.2. DESCRIPCIÓN

El framework SOFIA utiliza el modelo de implementación MVC (Modelo-View-Controller) para desarrollo en J2EE. La meta del framework es separar el código Java de la capa de presentación, para esto SOFIA hace uso de librerías de tags (etiquetas) que son usadas en las páginas JSP ya sea solas o en conjunto con código HTML. El framework maneja el concepto de controladores que "escuchan" los eventos que se dan en las páginas. Uno o más controladores pueden manejar los diferentes eventos que se disparan en una página, y responder acorde a lo requerido.

La parte que se conoce como modelo es implementada a través de los de los llamados datasouces. El front-end (capa de presentación) hace uso de estas instancias para identificar los datos que se desea usar y mostrar. Un datasource interactúa directamente con la base de datos manejando transparentemente sentencias SQL.

El concepto de MVC no es nuevo, pero SOFIA ha tratado de sacarle provecho mediante el uso de diferentes herramientas de desarrollo y extensiones adicionales. Por ejemplo SOFIA utiliza Macromedia Dreamweaver para el diseño de la capa de presentación incluyéndole una extensión adicional (plug-in) que inserta menús para crear la interface gráfica de usuario. Los componentes son incluidos en la página simplemente arrastrándoles a la posición deseada (drag-and-drop). La extensión utiliza un servlet traductor del framework que convierte los tags en código HTML lo que permite tener una pre-vista de la aplicación en Dreamweaver. Este servlet traductor puede estar situado en cualquier parte, permitiendo instalar únicamente la

extensión de SOFIA en Dreamweaver en una computadora asignada simplemente al diseño de la interface de usuario.

SOFIA provee a Dreamweaver de una paleta con más de 60 componentes para la creación de la capa de presentación. Son botones que permiten desde la creación de objetos que puede tener un formulario web hasta la creación de árboles o menús de navegación. Existen otros componentes no visuales como por ejemplo datastores, validadotes, etc. Cada componente pueden integrarse a la página mediante edición de código o arrastrándolos desde los paneles de herramientas.

SOFIA adicionalmente agrega una extensión en Eclipse para utilizarlo como generador de código (útil para la creación de los controladores y de la capa intermedia). SOFIA integra estas herramientas junto con servidores J2EE (Apache Tomcat, BEA WebLogic, IBM WebSphere). Mediante el uso de Eclipse se puede ejecutar el proyecto directamente, además permite levantar y parar el servidor a voluntad.

SOFIA integra tanto en Dreamweaver como en Eclipse un asistente de código, un inspector de propiedades o un revisor de errores sintácticos.

Los requerimientos para el funcionamiento de SOFIA a primera vista pueden considerarse extensos y complicados (JDK, Eclipse, Dreamweaver, Tomcat, y MySQL), pero en la practica no es así, además que pero la instalación es bastante sencilla.

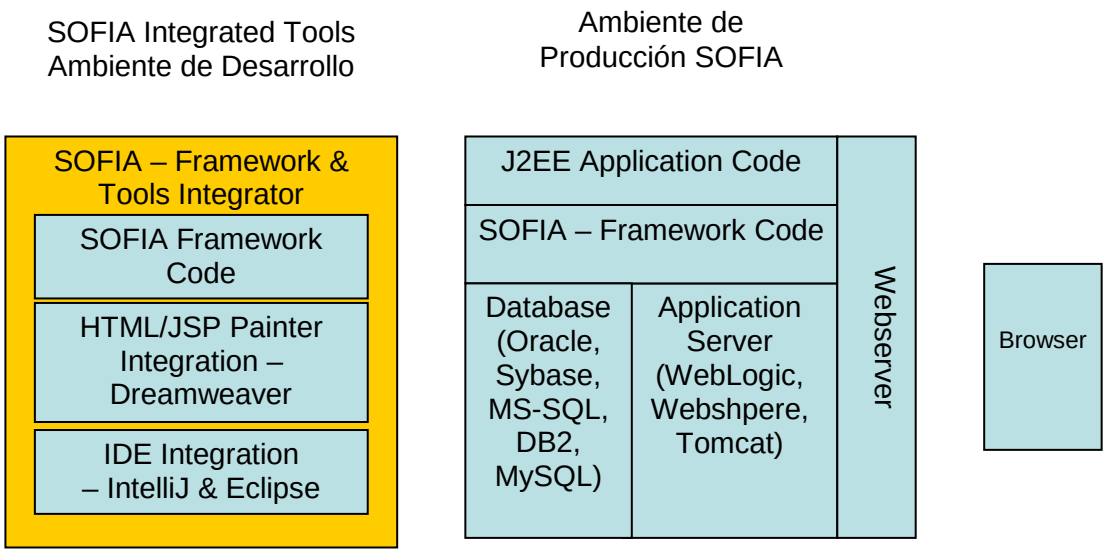


Figura 3.1: Arquitectura de Desarrollo SOFiA
Fuente: Salmon SOFIA Overview Presentation, Salmon LLC Site
Autor: Dan Dubinsky, SOFIA Lead Architect

SOFIA integra productos de diferentes compañías en ambiente de desarrollo integrado para la generación de aplicaciones MVC, y sin necesidad de escribir líneas de código para controlar aspectos básicos de una aplicación ya que cada componente tiene integrada su funcionalidad y todo se lo hace a través de ventanas de ayuda.

La creación de la interface gráfica mediante SOFIA se puede resumir en tres pasos: a) se especifican el controlador para manejo de eventos; b) se crean los datasources para acceso a la base de datos; c) se insertan los componentes gráficos que utilizarán los datasources para mostrar los datos; d) y finalmente crea el controlador de la página para manejar los eventos. Las tres primeras etapas se las realiza en Dreamweaver y la última en Eclipse. Así las etapas de desarrollo están bien definidas. Dentro de un equipo de trabajo se puede asignar personas que se dediquen únicamente a la creación de la interface gráfica y otras que manejen el back-end de la aplicación.

Las pruebas y depuración de la aplicación se la hace median Eclipse utilizando su capacidad de ejecución paso a paso y definición de *breakpoints*. Estos cambios pueden realizarse en caliente, es decir no es necesario reiniciar el servidor para hacer cambios.

3.3. ARQUITECTURA

3.3.1. MVC (Model- View-Controller)

El diseño de una aplicación de acuerdo con esta arquitectura de capas presenta la ventaja de producir código en donde la modificación de uno de sus componentes (por ejemplo la interface de usuario) no requiere modificación de ninguno elemento de otra capa. Asimismo, si por cualquier razón se migra el repositorio de datos (cambio de base de datos), únicamente sería necesario modificar la capa de datos.

Uno de los patrones de diseño más utilizados en Java es el MVC (Modelo-Vista-Controlador), cuyo principal objetivo es separar la presentación de la lógica de negocio. Utilizando MVC, la aplicación se convierte en una arquitectura completamente estructurada que divide perfectamente lógica de negocio (Model), presentación (View) y control de flujo de la aplicación (Controller). MVC es un patrón de diseño creado originariamente para el lenguaje SmallTalk[32].

El *controlador* define el comportamiento de una aplicación, es el encargado de manejar los requerimientos del usuario y seleccionar la vista adecuada para la presentación de la información requerida. Además de redirigir o asignar una petición a cada modelo, el controlador debe poseer de algún modo un "mapa" de correspondencias entre peticiones y acciones.

[32] Palos, Juan Antonio. *El API Struts*. Disponible en: <http://www.programacion.com/java/tutorial/struts/1/>

El *modelo* sería la acción que responde a una petición, representa la lógica de negocio y los datos de este; y posee las operaciones para acceso y modificación de dichos datos. El modelo notifica a las vistas cuando los datos han cambiado y provee al controlador la habilidad de acceder a estos. Una vez que el modelo realiza las operaciones necesarias el flujo retorna al controlador y éste devuelve los resultados a una *vista*. La vista muestra los datos proporcionados por el modelo[33].

Vemos las diferencias que suponen los modelos de aplicaciones convencionales cliente/servidor de 2 capas donde se tiene una entrada con parámetros que ingresan (INPUT), éstos se procesan y se muestra el resultado (OUTPUT):

Figura 3.2: Esquema básico de un programa convencional
Fuente: STRUTS:Implementación del patrón MVC en aplicaciones Web
Autor: Pello Xabier Altadill Izura; Javahispano.com

En el caso del patrón MVC el procesamiento se lleva a cabo entre sus tres componentes. El controlador recibe una orden y decide que modelo debe llevar a cabo el requerimiento, una vez que el modelo termina sus operaciones devuelve el flujo al controlador y éste envía el resultado a la capa de presentación:

[33] Gándara, Juan Simal. *Desarrollo de Aplicaciones con tecnología Internet*. Disponible en: <http://www.soluziona.es/htdocs/areas/consultoria>

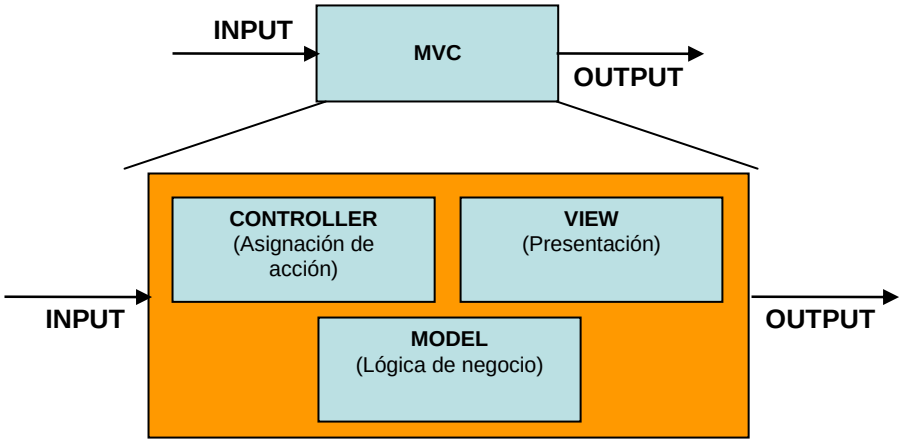


Figura 3.3: Esquema del modelo MVC
Fuente: STRUTS: Implementación del patrón MVC en aplicaciones Web
Autor: Pello Xabier Altadill Izura; Javahispano.com

El controlador en cierta forma debe tener un registro de la relación entre órdenes que le pueden llegar y la lógica de negocio a la que le corresponde actuar (por ejemplo se podría comparar con una operadora de teléfono que recibe una petición y une dos líneas). Existen muchas ventajas al utilizar este modelo ya que se separa totalmente la lógica de negocio y la presentación y permite varias opciones útiles como multilenguaje o temas de presentación, y sin alterar la lógica.

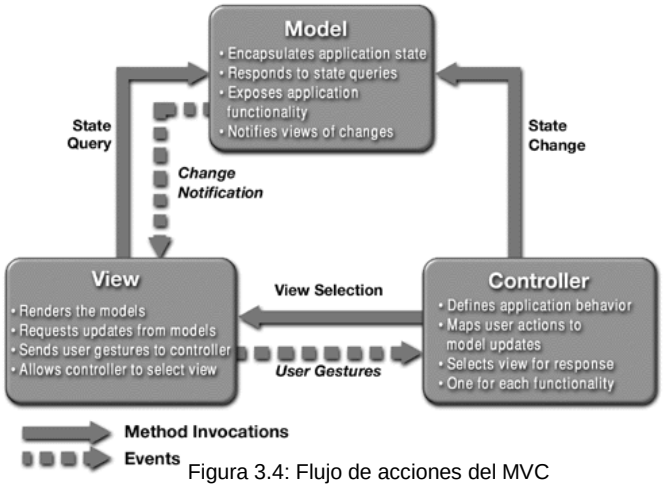


Figura 3.4: Flujo de acciones del MVC
Fuente: java.sun.com
Autor: Sun Microsystems

La separación de las capas es fundamental para el desarrollo de arquitecturas consistentes, reutilizables y de fácil mantener; lo que al final resulta en un ahorro de tiempo en desarrollo para posteriores proyectos.

En la práctica, cuando se ejecuta una aplicación, todas las capas o bloques del patrón MVC actúan en conjunto:

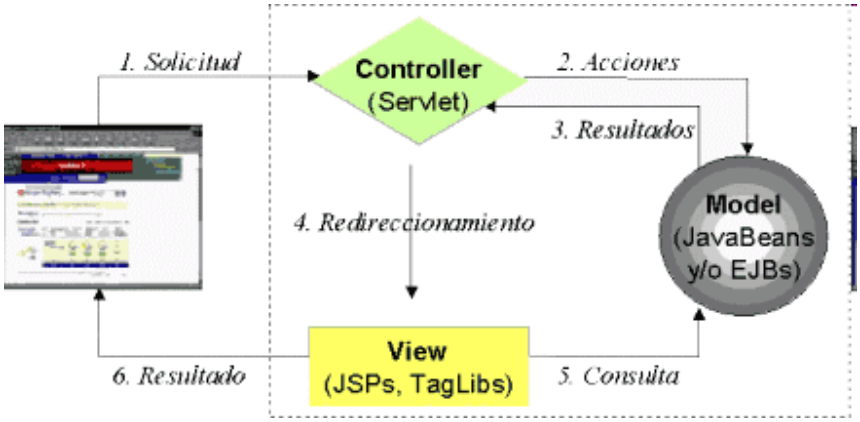


Figura 3.5: Funciones del controlador
Fuente: Manual Básico de Struts; <http://www.programacion.com/java/tutorial/joa_struts>
Autor: Javier Antoniucci

El navegador del usuario, en su petición HTTP, genera una solicitud que es atendida por el controlador. Éste analiza sus entradas y llama a los objetos correspondientes del modelo. El modelo se encarga de ejecutar la solicitud y generar los resultados que mostrarán posteriormente las vistas.

El controlador recibe todas las peticiones realizadas a la aplicación. Cada petición se identifica mediante un parámetro. En base a esta identificación, el controlador decide qué objeto u objetos de negocio (modelo) debe ejecutar para resolver la petición.

Tras la ejecución de los objetos de negocio, y en función del resultado devuelto por estos, el controlador determina qué JSP se usará para visualizar el resultado, generando una redirección que concluirá con la generación del código HTML de la página[34].

Los modelos que son los que implementan la lógica de negocio poseen las siguientes funciones dentro de una aplicación:

- Realizar la validación de los datos introducidos por el usuario tanto sintáctica (numérico, fecha, etc.) como lógica (importe menor que saldo, etc.).
- Ejecutar la petición realizada por el usuario, para lo que se valen de consultas a bases de datos u otras fuentes.
- Generar los objetos que utilizarán las vistas para mostrar los resultados obtenidos.
- Garantizar la navegación y el flujo de pantallas correcto.

Normalmente el flujo de la aplicación está dirigido por un controlador central. El controlador delega solicitudes a un manejador apropiado (subcontrolador). Los manejadores están unidos directamente a un modelo, y cada manejador actúa como un adaptador entre la solicitud y el modelo. Esto proporciona una estructura jerárquica de control de páginas y disminuye la duplicación de código, lo que hace más fácil de crear y de mantener una aplicación.

[34] Antoniucci, Javier. *Manual Básico de Struts*. Disponible en: <http://www.programacion.com/java/tutorial/joa_struts>

Las principales ventajas y desventajas del patrón MVC son:

Ventajas	Facilidad de desarrollo y reducción de tiempos gracias a la paralelización de tareas.
	Facilita la especialización de perfiles de programación (cada uno de los diferentes bloques es desarrollado por distintos equipos en función de sus especialización).
	Aumenta en gran medida el nivel de reusabilidad de código. Facilita una evolución continuada de los sistemas, ya que un cambio en un sistema afectará a uno o más componentes pero nunca afectará significativamente al núcleo de la aplicación
	Soporte para múltiples vistas puesto que la vista se separa del modelo y no hay ninguna dependencia directa entre estas, la interface de usuario puede mostrar múltiples vistas de los mismos datos al mismo tiempo.
	Mayor soporte a los cambios, ya que los requisitos de interface tienden a cambiar más rápidamente que las reglas de negocio. Puesto que el modelo no depende de las vistas, la adición de nuevos tipos de vista al sistema generalmente no afectan al modelo.
Desventajas	Complejidad. El patrón MVC introduce nuevos niveles de interacción y por tanto incrementa la complejidad de la solución. También incrementa la dificultad de manejar eventos de la interface de usuario, lo que puede complicar la depuración.
	Coste de actualizaciones frecuentes, ya que si el modelo sufre cambios frecuentes, podría sobrecargar a las vistas con solicitudes de actualización. Algunas vistas pueden tardar un tiempo en redibujarse, por tanto la vista puede estar desactualizada. Esto se puede solucionar en parte si el diseñador piensa en las vistas cuando codifica el modelo y, por ejemplo, envía varias actualizaciones juntas en una sola notificación a la vista.

Cuadro 3.1: Ventajas y desventajas del patrón MVC
Autor: Edison Hernández

3.3.2. J2EE

Al basarse en la plataforma J2EE, SOFIA aprovecha de muchas de sus ventajas y trata simplificar el uso de la plataforma. J2EE es un conjunto de especificaciones donde no existe un framework o base de desarrollo de la cual se pueda partir. Además el código puede ser difícil de complicado de mantener, por lo que SOFIA trata de disminuir estas falencias propias de J2EE.

Para más información referirse al Capítulo 1 de este documento.

3.3.3. Framework

Un framework o marco de trabajo (también llamado entorno de trabajo) tiene por intención facilitar el desarrollo de aplicaciones. Brinda pautas para que el desarrollador no comience desde cero la creación de una aplicación, sino que tenga una base de desarrollo y no tenga que preocuparse en la creación de cosas triviales (y de forma) que no tienen nada que ver con la aplicación en sí. Un framework normalmente ofrece un conjunto de herramientas, ambientes o librerías para esto; lo que evita que el programador construya algo que ya esta construido con anterioridad. La meta principal de Struts es separar la lógica de presentación de la lógica del negocio.

Un framework es la extensión de un lenguaje mediante una o más jerarquías de clases que implementan una funcionalidad y que (opcionalmente) pueden ser extendidas.

3.3.3.1. Otros Frameworks para J2EE (Struts)

Struts se lo lanzó en Mayo del 2000 por Craig R. McClanahan para proporcionar un marco de trabajo MVC estándar para Java. En Julio del 2001, se liberó Struts 1.0. En la actualidad Struts se distribuye bajo la licencia de la *Apache Software Foundation*. El código tiene copyright pero es gratuito para usarlo en cualquier aplicación[35].

Struts es un framework para desarrollo de aplicaciones web en Java que implementa el modelo Model-View-Controller. Realmente lo que provee es un conjunto de clases

[35] Palos, Juan Antonio. *El API Struts*. Disponible en: <http://www.programacion.com/java/tutorial/struts/>

y librerías de etiquetas (tags-libs) que conforman el *controlador*, la integración con el *modelo* (o lógica de negocio) y facilita de alguna manera la construcción de *vistas*.

El controlador ya se encuentra implementado por Struts. Las acciones que se ejecutan sobre el modelo de objetos de negocio se implementan basándose en clases predefinidas por el framework. La generación de la interface se soporta mediante un conjunto de Tags predefinidos por Struts cuyo objetivo es evitar el uso de Scriplets, lo cual genera ventajas de mantenimiento y de rendimiento. Además permite el desarrollo de las diferentes capas en paralelo así como la reutilización de código, soporte de múltiples interface de usuario y de múltiples idiomas[36].

Entre las características que ofrece Struts para el desarrollo se destacan:

ActionForms	Formularios con manejo de eventos, existen formularios tanto estáticos como dinámicos.
Tiles	Utilizados para creación y trabajo con plantillas (páginas Web).
Actions	Acciones únicas (eventos) y de grupo.
ActionMappings	Permiten mapear las características de las acciones en un fichero externo XML
ActionErrors	Para devolver mensajes de error en la validación de formularios ActionForms
Contenedor	Utilizados para guardar toda la información necesaria de sesión
Struts Tags	Librería de etiquetas que ahorra código
Servicios	Como por ejemplo para conexión con la BD mediante JDBC.

Cuadro 3.2: Componentes importantes de Struts
Fuente: Apache Software Fundation; Struts Tutorial
Autor: Edison Hernández

Las aplicaciones Struts residen en un contenedor Web dentro de un servidor de aplicaciones y pueden hacer uso de servicios proporcionados por este contenedor (manejo de peticiones HTTP o HTTPS) lo que permite al desarrollador olvidarse de

[36] Apache Software Fundation. *Struts Tutorial*. Disponible en: <http://www.apache.org>

la lógica de negocio. Desde el punto de vista de la arquitectura MVC, las clases base que proporciona Struts son:

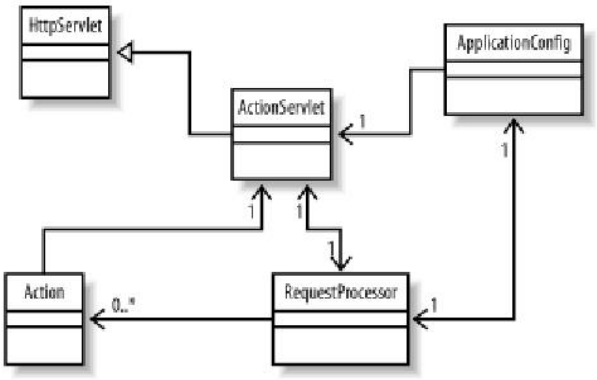


Figura 3.6: Clases que proporciona Struts
Fuente: www.adictosaltrabajo.com
Autor: Enrique Medina Montenegro

La clase *ActionServlet* es responsable del empaquetado y enrutado del tráfico HTTP hacia el manejador apropiado dentro del framework. No es abstracta, y por tanto se puede utilizar directamente en cualquier aplicación.

La clase *RequestProcessor* permite desacoplar el proceso de petición (request process) del ActionServlet y así poder modificar cómo se procesa la petición.

La clase *Action* independiza la petición del cliente con respecto al modelo de negocio. Es una extensión del componente de control (controlador) y permite llevar a cabo funciones como autorización, logging, o validación de sesión antes de invocar la lógica de negocio. Su método más importante es el *ActionForward*.

La clase *ActionMapping* representa una acción de mapeado que se define en el fichero de configuración de Struts. Indica al controlador que instancia de Action se debe invocar en cada petición.

La clase *ActionForward* representa el destino al cual el controlador debe enviar el control una vez que un *Action* se ha completado[37].

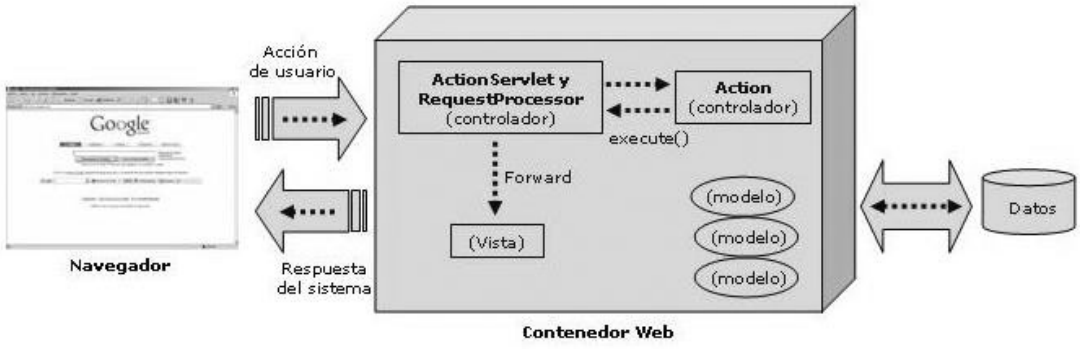


Figura 3.7: Como trabaja Struts
Fuente: www.adictosaltrabajo.com
Autor: Enrique Medina Montenegro

La idea es que Struts reciba la información en forma de vistas pero no sepa cómo se han creado esas vistas. Para ello es necesario crear un servicio perteneciente a la capa de negocio, que servirá de nexo con la capa de control. Por el otro lado, la capa de negocio sólo sabrá que puede recibir peticiones del servicio de acceso a BD a través de JDBC, sin importarle ni quién los pide ni quién los usará a continuación.

Con respecto a la capa de vista, los elementos de los que hace uso Struts son HTML y ActionForms que son componentes que permiten pasar datos entre el usuario y la capa de negocio; y todo esto mediante JavaServer Pages y Struts Tags (librería de etiquetas).[38]

[37] Altadill Izura, Pello Xabier. *Struts: Implementación del patrón MVC en aplicaciones Web*.
[38] Medina Montenegro, Enrique. *Como trabaja Struts*. Disponible en: <www.adictosaltrabajo.com>

En este entorno, se debe invocar una acción o aplicación que debe estar mapeada en Struts, tal acción se corresponderá con una clase Java heredera de la clase *Action*. El mapeo de acciones y clases se especifica en el fichero *struts-config.xml* que es de importancia vital para Struts. Ahí se especifican todas las relaciones entre acciones y clases, formularios y clases, acciones y JSPs de presentación, que globalmente conforman el "mapa" de la aplicación.

El controlador Struts une y enruta solicitudes HTTP a otros objetos del framework, incluyendo JavaServer Pages y subclases. Una vez inicializado, el controlador analiza un fichero de configuración de recursos que define entre otras cosas los *ActionMapping* para una aplicación. El controlador usa estos mapeos para convertir las solicitudes HTTP en acciones de aplicación.

Para la aplicación más simple, un objeto *Action* podría manejar la lógica de negocio asociada con una solicitud; sin embargo, en la mayoría de los casos, un objeto *Action* debería llamar a otro objeto, normalmente un JavaBean, para realizar la lógica de negocio real. Esto permite al objeto *Action* enfocarse en el manejo de errores y el control de flujo, en vez de en la lógica del negocio. Para permitir su reutilización en otras plataformas, los JavaBeans de lógica de negocio no deberían referirse a ningún objeto de vista. El objeto *Action* debería traducir los detalles necesarios de la solicitud HTTP y pasarlos a los *Beans* de la lógica del negocio como variables normales de Java.

Por ejemplo, en una aplicación de base de datos un *Bean* de lógica de negocio conectaría y consultaría la base de datos. Este *Bean* devolvería el resultado al objeto *Action*. El objeto *Action* almacenaría el resultado en un *Bean* formulario en la solicitud. La página JSP mostraría el resultado en un formulario HTML. Ni el objeto *Action* ni la página JSP necesitan saber de dónde viene le resultado. Sólo necesitan saber cómo empaquetarlo y mostrarlo.

Si quisiéramos crear la aplicación Struts más simple posible, por ejemplo una página con un saludo, debiéramos hacer lo siguiente:

1. Una página JSP (la presentación)
2. Una clase Action (componente del controlador)
3. La clase Action debe definirse en el struts-config.xml correctamente

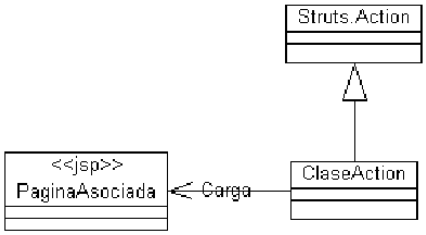


Figura 3.8: Funciones del controlador
Fuente: Javahispano.com

Como se puede ver no se tendría más que dos ficheros, y una clase que hereda de Struts. La clase *Action* se configura en el *strtus-config.xml* y se convierte en parte del controlador. Cuando la aplicación recibe una petición, Struts decidirá que debe cargar en esa clase y a través de ella cargará la página JSP[39].

[39] Altadill Izura, Pello Xavier. *Struts: Implementación del patrón MVC en aplicaciones Web*.

3.3.3.2. Comparación con Struts

SOFIA en comparación con Struts es considerada una herramienta mucho más fácil de utilizar debido a que Struts no se integra directamente con IDEs o herramientas HTML para creación de interfaces. Struts es definido como un framework de bajo nivel que requiere escribir muchas más líneas de código, además no tiene integración directa con base de datos.

Struts necesitaría de varias de herramientas trabajando juntas para simular todas las características que ofrece SOFIA:[40]

- Struts +
- J-Expresso +
- Muchas extensiones adicionales de Struts (indexed tags, row tag, grid tag, multi-controller, etc.) las cuales deben ser integradas manualmente por el desarrollador.

Pero aun con todas estas herramientas Struts no ofrece un ambiente GUI tan flexible como el de SOFIA.

[40] Salmon LLC. *The SOFIA Philosophy*. Disponible en: <http://www.salmonllc.com/website/Jsp/vanity/SofiaPhilosophy.jsp>

A continuación se presenta un cuadro existente en la documentación de SOFIA que compara sus cualidades con Struts:

Características	SOFIA	STRUTS
Arquitectura	JSP/J2EE/ Framework Model-View-Controller para aplicaciones web.	JSP/J2EE/ Framework Model-View-Controller para aplicaciones web.
Creación de Vistas	Vistas pueden ser creadas usando JSP y HTML + Salmon Custom Tags o código Java usando componentes AWT.	Vistas pueden ser creadas utilizando JSO y HTML + Struts Custom Tags.
Desarrollo de interfaces gráficas	Gracias a la extensión adicional en Dreamweaver se soporte para los *Salmon Custom Tags Library*. La creación de la GUI se puede hacer a través de edición de código agregando los componentes *drag-and-drop* en vistas gráficas.	No posee generador de GUI integrado. No existe integración con Dreamweaver. GUI pueden ser creadas en *Object Adventure's Object Assembler* (requiere JBuilder). El *Object Assembler* es más un generador de código que un generador de interfaces gráficas.
Componentes visuales inteligentes	Posee matrices para muestra integradas con botones de paginación o capacidad de ordenamiento, árboles de navegación, barras de navegación, calendarios, formularios de búsquedas, etc.	No posee componentes visuales integrados. Extensiones adicionales permiten generación de componentes visuales básicos para interfaces.
Capacidad para crear custom tags (Agregar nuevos componentes visuales)	Componentes AWT pueden ser combinados o extendidos para la creación de nuevas etiquetas personalizadas (custom tags) mediante la utilización de clases de SOFIA.	Etiquetas personalizadas pueden ser creadas desde cero.
Código Java mezclado en JSP	No es requerido código Java en páginas JSP. No es requerida lógica de la aplicación en la vista. Se puede manipular la vista mediante código Java dentro de un controlador o cambiando datos en el modelo. Para mantener encapsulamiento, la lógica puede ser incluida en la vista mediante "property expression" que son expresiones para manipulación de propiedades (color, fuente, visibilidad, habilitado) para componentes GUI.	No es requerido código Java en páginas JSP. Lógica para el control de la presentación es codificada en la vista por medio de los llamados "Struts Logic tags" El controlador tiene un control únicamente indirecto sobre la vista mediante ítems en la clase http Session.
Internacionalización	Soportado mediante el manejo archivos planos o bases de datos.	Soportado
Creación del los controladores	Los controladores son extendidos desde la clase "Salmon JSPController".	Controladores son extendidos de la clase "Struts Action Class".

Acceso de controladores a formularios	El acceso a la información de formularios por medio del controlador es hecho automáticamente por el framework y no requiere escribir código adicional. Valores ingresados en formularios son vinculados a componentes de la GUI instanciados en el controlador. Estos valores también pueden ser transferidos directamente al modelo usando "bound GUI components".	Acceso a valores de formularios es hecho por el Bean *ActionForm*. El programador tiene que escribir este Bean manualmente para después vincularlo a los componentes de los formularios poniéndoles el mismo nombre que constan en los el Bean para utilizar los métodos get y set. La transferencia de datos desde el *ActionForm* hacia el modelo debe ser hecho manualmente mediante llamadas a los métodos get y set.
Manejo de eventos en el controlador	Es usado un "escuchador" de eventos y validaciones en el modelo. Estos "escuchadores" son registrados median llamadas a métodos Java. Una vez registrado, los eventos son automáticamente vinculados al componente que está interesado en procesarlo.	El manejo es hecho mediante el archivo *config.xml*. Aquí se especifica que JSPs deben invocar las acciones. La clase *Struts Action* tiene un evento que es llamado cuando una página es requerida. Si múltiples componentes o componentes que no son acciones necesitan ser manejados como eventos, el programador debe hacerlo manualmente
Soporte RDBMS	SOFIA provee un *pooler* de conexiones y componentes para abstracción de la base de datos. Además soporta transacciones JDBC.	No soporta bases de datos relacionales de manera directa. Existen extensiones adicionales como el framework JExpresso para realizar esto.
Captura de datos en el servidor de aplicaciones	No es incluida directamente. La captura de datos en el servidor de aplicaciones mejora el desempeño pero dificulta la interacción con la base de datos.	No tiene soporte de este tipo
Soporte para EJB	Soporte de manera indirecta. El programador debe vincular manualmente los componentes datastores a los EBJ.	No soporta EJB de manera directa. Mediante extensiones adicionales se puede crear Entity Beans.

Cuadro 3.3: Metas de SOFIA
Fuente: Struts vs The Salmon Open Framework for Internet Applications
Autor: Salmon LLC Site

3.4. REQUERIMIENTOS

Actualmente SOFIA se encuentra en la versión 2.3 lanzada a fines de 2004. Todos los requerimientos especificados se basan en la documentación que ofrece SOFIA para esta versión.

3.4.1. Software

Los requerimientos de SOFIA son los siguientes:

Sistema Operativo	SOFIA trabaja únicamente bajo el sistema Operativo Windows. Las versiones recomendadas para su ejecución son Windows NT 4.0, Windows 2000 o Windows XP. Otras versiones de Windows no son recomendadas debido a posibles inestabilidades en el sistema.
Paquete de desarrollo	JDK (Java Development Kit) es el paquete de desarrollo requerido para Java. Puede utilizarse la versión 1.4.x. Este puede ser bajado de del sitio oficial de Sun Microsystems. Hay que asegurarse de bajarse la versión completa SDK (System Development Kit) y no simplemente el JRE. La versión del JDK 1.4.2 es perfectamente compatible con SOFIA. JDK es parte del la plataforma de desarrollo J2SE de Java. J2SE provee un conjunto de librerías y ambientes de desarrollo para aplicaciones en Java. Junto con instalador de JDK viene la versión del JRE (Java RunTime Environment) para ejecución de aplicaciones Java del lado del cliente.
Base de datos	En realidad SOFIA puede trabajar con casi cualquier base de datos pero es recomendada MySQL. MySQL es una de las bases de datos más populares en la actualidad debido a si rapidez y facilidad. Ocupa muy pocos recursos tanto de procesador como de memoria. SOFIA puede trabajar con cualquier versión de MySQL ya sea 3.23 o 4.x. Esta base de datos es gratuita y se distribuye bajo la licencia GLP.
Contenedor de Servlets y JSPs	Igualmente SOFIA permite trabajar con varios contenedores de servlets pero el más utilizado es Tomcat. Tomcat es un contenedor de Servlets y JSPs que viene integrado con el servidor web Apache. Es una herramienta gratuita que se encuentra bajo la licencia Apache Software. SOFIA soporta la versión 5.0 de Tomcat, pero en ocasiones pueden presentar problemas de compatibilidad.

IDE tool	Para la instalación de SOFIA es necesario tener uno de los dos IDEs (Eclipse o IntelliJ's IDEA) recomendados. Son muy parecidos en características pero Eclipse es gratuito. Eclipse es un IDE para desarrollo de aplicaciones en Java que trata de facilitar el desarrollo de aplicaciones y la escritura de código. La versión 2.3 de SOFIA soporta hasta la versión Eclipse 3.0 o IntelliJ 4.0. Eclipse es el IDE que se utilizará para el desarrollo de esta tesis y puede se bajado de la página oficial **www.eclipse.com**.
GUI Tool	Macromedia Dreamweaver es el editor de desarrollo Web más utilizado a nivel profesional para la creación de páginas para internet. Su amplio abanico de herramientas permite crear desde la más simple página web personal hasta el sitio web más completo y complejo. Es la herramienta utilizada para la creación de las interfaces de usuario mediante JSP con un ambiente de desarrollo muy gráfico e intuitivo. Extensiones adicionales son adicionadas a Dreamweaver para integración de SOFIA. La última versión es la MX 2004. Esta es una herramienta comercial por lo que tiene un costo, pero se puede bajar una versión de prueba de la página oficial de Macromedia en ***www.macromedia.com***.
Instalador de SOFIA	La última versión de SOFIA es la 2.3. Puede ser bajada de la página oficial de SOFIA o en la comunidad de desarrollo SourceForge.

Cuadro 3.4: Requerimientos de Software
Fuente: Documentación de SOFIA

A pesar de requerir únicamente las herramientas listadas anteriormente, existen otras opciones que son compatibles con SOFIA. A continuación se especifican todas las herramientas compatibles:

Tipo	Nombre
IDE	IntelliJ 2.5/2.6/3.0 Eclipse 3.0
HTML Design	Dreamweaver 4.0/MX
Database	Sybase Oracle MS-SQL DB2 Mainframe (VSE & MVS) DB2 Desktop MySQL
Application Server	Tomcat v5.0/4.0/4.1/3.x WebLogic v6.1 Websphere v3.5/4.0/5.0 Resin v2.1.6 Silverstream v3.7.4
Web Servers	Apache IIS IBM HTTP Server

Content Management	Fatwire Update Engine 5.0
Document Formats	PDFs Word Documents

Cuadro 3.5: Productos compatibles con SOFIA
Fuente: Documentación de SOFIA

3.4.2. Hardware

Si se quiere tener un rendimiento aceptable es conocido que para el desarrollo de aplicaciones en Java las características del equipo deben ser elevadas. Además hay que recordar que SOFIA es un conjunto de herramientas que deben ejecutándose al mismo tiempo.

Para el correcto funcionamiento de SOFIA se considera los siguientes requerimientos:

Memoria	Mínimo: 256 MB Recomendado: 512 MB
Disco Duro	Al menos 80 MB de espacio en disco para la instalación
Procesador	Mínimo: Pentium III de 450 MHz. Recomendado: Pentium IV de 1.6 GHz.

Cuadro 3.6: Requerimientos de hardware
Fuente: Documentación de SOFIA

Según experiencias personales, los requerimientos de hardware para SOFIA no debería ser menor a los 512 MB de memoria con un procesador Pentium 4 no menor a los 1.6 Ghz. Esto es porque Eclipse y Dreamweaver están frecuentemente abiertos simultáneamente y ocupan bastantes recursos del sistema. Igualmente en la ejecución de las aplicaciones el levantar y bajar el servidor de aplicaciones (Tomcat) puede ocasionar caídas en el rendimiento del sistema si no se toman las precauciones necesarias.

3.5. ANÁLISIS ECONÓMICO

3.5.1. Licencias

SOFIA es una herramienta gratuita. Ha sido lanzado bajo 2 tipos de licencias. Versiones no registradas de SOFIA se encuentran bajo la licencia GNU GPL, esto significa que proyectos que son creados con SOFIA tienen que ser lanzados también bajo la licencia GNU GPL como código abierto. Una vez que SOFIA sea registrado, se podrán crear proyectos usando la licencia de Salmon, esta licencia permite la distribución y comercialización de la aplicación creada sin tener que abrir el código.

La última versión de SOFIA se encuentra bajo las versiones "GNU General Public License version 2" y "The Salmon Software License, Version 1.0". La licencia de Salmon específica que la redistribución o uso del código binario, con o sin modificación, está cubierta tanto por las licencias GNU GPL y Salmon License.

Al programa no le cubre ninguna garantía aparte de la exigida por las leyes. El riesgo que puede existir al utilizar esta herramienta depende del desarrollador y de nadie más[41].

Con respecto a los otros componentes de SOFIA, Eclipse también es una herramienta gratuita. Como se explica en el Capítulo 2 de este documento, se distribuye actualmente bajo la licencia CPL (Common Public License) versión 1.0 de IBM.

[41] Salmon LLC. *Salmon Software License Documentation*

Esta licencia permite que Eclipse pueda extenderse mediante la inclusión de plug-ins propietarios o ser usado como base para la creación de nuevas herramientas, y, tras reempaquetarse y compilarse el código resultante, el producto final puede venderse de forma comercial, manteniéndose público el código de Eclipse utilizado, pero sin la obligación de poner a disposición del público el nuevo código añadido.[42]

El paquete de desarrollo de Java (JDK) es también gratuito. La plataforma de desarrollo J2SE (Java 2 Standart Edition) junto con el JRE (Java Runtime Environment) son libres de descargarlas y usarlas con fines comerciales. También son libres para distribución si se lo hace con mejoras adicionales.

MySQL en principio es gratuito. Tiene 2 políticas de licenciamiento: La licencia comercial (Commercial License) que permite proveer licencias comerciales a los clientes o distribuir aplicaciones con MySQL dentro de su organización; este tipo de licencia es para organizaciones que no tienen la intención de liberar los códigos de sus aplicaciones o de distribuir su software gratuitamente. El segundo tipo de licencia es de código abierto (Open Source License) la cual puede ser utilizada para el desarrollo de aplicaciones open-source de distribución libre.[43]

Tomcat es gratuito y de código libre. Al ser parte del la fundación Apache, Tomcat puede ser usada por cualquier persona y esta puede realizar las modificaciones que

[42] Eclipse Project. *Eclipse Tutorial Documentation*
[43] MySQL AB. *MySQL Licensing Policy*. Disponible en: <http://www.mysql.com/company/legal/licensing/>

desee al código fuente. También Tomcat viene con una licencia para uso comercial identificada como ASF tanto para código fuente como para ejecutables[44].

Dreamweaver es la única herramienta que tiene un costo real y que su licencia es únicamente comercial. Existe versiones de prueba pero estas solo pueden ser usadas por tiempo limitado. Una organización debe comprar la licencia para poder utilizarla.

3.5.2. Costos

La mayoría de las herramientas necesarias para la utilización de SOFIA son gratuitas. A excepción de Dreamweaver, tanto SOFIA como JDK, Tomcat, MySQL y Eclipse se los puede adquirir y utilizar sin ningún recargo.

Dreamweaver en su versión última (MX 2004) tiene un costo oficial de $399 en Estados Unidos. Este costo es en la página oficial de Macromedia pero puede variar en algún grado dependiendo del lugar en el que se adquiere. Dreamweaver es una de las mejores herramientas para el diseño web y es la más utilizada entre los diseñadores por lo que es comprensible su costo.

[44] The Apache Jakarta Tomcat. *Tomcat F.A.Q.* Disponible en: <http://jakarta.apache.org/tomcat/faq/>

3.6. INSTALACIÓN Y CONFIGURACIÓN

En primer paso para instalar SOFIA es instalar el SDK de Java de acuerdo a lo especificado en la sección de requerimientos. Este instalador es un ejecutable que guía al usuario paso a paso durante el proceso.

Posteriormente es necesario instalar el IDE. En nuestro caso se va a utilizar Eclipse. Para su instalación es necesario obtener un archivo empaquetado y extraerlo en el disco duro. Solo es necesario ejecutarlo para que la instalación se complete. Se recomienda especificar en las configuraciones de Eclipse que se usará la versión 1.4 del SDK y no simplemente con el JRE.

Después se debe instalar Macromedia Dreamweaver. Esto se lo hace a través de un ejecutable. Se recomienda asegurarse de ejecutar por lo menos una vez Eclipse y Dreamweaver antes de continuar.

Finalmente se puede instalar SOFIA. Para ello existe un instalador que se ejecuta y configura los componentes a ser utilizados:

1. Inicialmente instala el framework de SOFIA y los códigos fuentes.
2. Opcionalmente se instala las extensiones (plug-ins) adicionales de Eclipse.
3. Después configura Tomcat como contenedor de servlets y JSP. Si Tomcat no está instalado se comenzará la instalación. No se debe instalar Tomcat como servicio ya que puede ocasionar conflictos con SOFIA

4. Opcionalmente instala las extensiones de Dreamweaver a través del Macromedia Extensión Manager.
5. Finalmente instala la base de datos y también documentación de SOFIA.

Adicionalmente se completa la instalación de la extensión de Eclipse de manualmente. Para esto hay que ejecutar SOFIA y realizar los siguientes pasos:

1. Activar el menú de SOFIA eligiendo "Windows/Customize/Other/Salmon Menu".
2. Después elegir "File/New Project" y seleccionar "Install/Upgrade Sofia Framework".
3. Posteriormente elegir "File/New Project" y seleccionar "Install/Upgrade Sofia Example Application".

Esos son todos los pasos necesarios para utilizar SOFIA. A partir de este instante el framework está a disposición del desarrollador[45].

[45] Salmon LLC. *SOFIA Version 2.2 Installation Guide*

3.7. CARACTERÍSTICAS

3.7.1. Antecedentes

El desarrollo de aplicaciones en SOFIA esta basado en la separación de las capas de la aplicación: el modelo, la vista y el controlador.

Inicialmente Salmon no consideró el lanzamiento al publico de SOFIA, ya que querían mantener la herramienta para uso interno de compañía, pero una vez creada la aplicación se dieron cuenta que el framework de desarrollo más utilizado para aplicaciones web en Java era Struts y era considerado muy difícil en comparación con SOFIA. Esta empresa consideraba a Struts difícil, tedioso, sin sentido y muy trabajoso. Salmon vio muchas de las funcionalidades de Struts en SOFIA al integrar un gran número de diferentes herramientas open-source pero sin un editor de GUI ya Struts lo hacia mediante algún editor de texto.

Posteriormente con la salida de Microsoft .Net Studio, a primera vista les dio la impresión de parecerse mucho a SOFIA y se fijaron que tenía mejor aspecto que muchas de las herramientas para Java existentes. Se dieron cuenta que si la comunidad Java no hacia algo para cambiar esto, Microsoft comenzaría a ganar más y más adeptos. Al no tener ningún deseo de convertirse en programadores Microsoft, Salmon decidió en lanzar SOFIA como open-source creyendo que esto ayudaría a surgir a Java como una plataforma más poderosa[46].

[46] Salmon LLC. *The SOFIA Philosophy*. Disponible en: <http://www.salmonllc.com/website/Jsp/vanity/SofiaPhilosophy.jsp>

Las metas principales de SOFIA son el:

Mejoramiento de la eficiencia del desarrollador	• Crear ambientes GUI permitiendo un proceso más rápido y más intuitivo. • Permitir al desarrollador concretarce en los requerimientos de la aplicación más altos sin tener que controlar cosas triviales de una aplicación • Rehusar componentes estables para incremento de velocidad y consistencia de desarrollo. • Proveer un punto de partida consistente y adecuado.
Proveer de flexibilidad al desarrollador	• Utilizar métodos como el drag-and-drop o código reutilizable. • Crear el código de la capa de presentación en HTML y JSP • Permitir que elementos de la interface de usuario sean creados en etapas de diseñados o de ejecución.
Mejorar el mantenimiento de las aplicaciones	• Código consistente y que soporte estándares de desarrollo. • Remover código Java dentro de las páginas JSP. • Reducir el número de líneas de código para un proyecto. • Disminuir el número de errores que se puedan presentar en etapas de ejecución.
Usar únicamente estándares abiertos	• Java • JSP

Cuadro 3.7: Metas de SOFIA
Fuente: Salmon SOFIA Overview Presentation, Salmon LLC Site
Autor: Dan Dubinsky, SOFIA Lead Architect

SOFIA es considerada una herramienta RAD (Rapad Application Development) debido a que facilita y sobre todo acelera la creación de aplicaciones empresariales mediante tareas básicas del sistema como:

Herramientas de construcción	Implementado en SOFIA?
Control de usuarios	√
Control de base de datos	√
Manejo de mensajes	√
Manejo de errores	√
Manejo de registros	√
Integración de aplicaciones	√
Generación de ayudas	√
Manejo de internacionalización	√
Manejo de estandarización	√

Cuadro 3.8: Tareas que realiza SOFIA
Fuente: Salmon SOFIA Documentation
Autor: Salmon LLC

3.7.2. Estándares y patrones

SOFIA hace uso de una serie de estándares, arquitecturas y patrones de desarrollo[47]:

Model View Controller (MVC): Mejora el mantenimiento de las aplicaciones. Se divide en 3 componentes: El modelo (Model) recoge los datos, la vista (View) muestra los datos, y el controlador (Controller) maneja la interacción con el usuario.

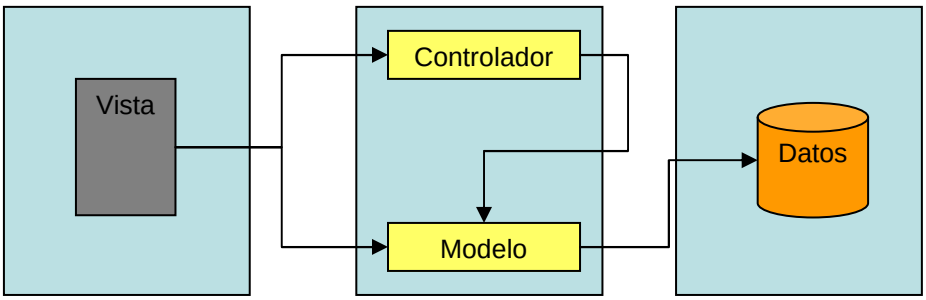


Figura 3.9: Flujo de datos del MVC en SOFIA
Fuente: STRUTS: Implementación del patrón MVC en aplicaciones Web
Autor: Pello Xabier Altadill Izura; Javahispano.com

Para más información referirse al inicio de este capítulo.

Smart GUI (Graphic User Interface) Components: Patrón para mejorar la eficiencia del desarrollador. SOFIA posee componentes simples (botones, campos de textos o tablas) pero también presenta componentes más complejos llamados "Ready to Go" (listos para usar) como Data Grids, calendarios, barras de menú o formularios de búsqueda. Estos GUI se vuelven inteligentes en la forma como pueden recordar su estado (datos ingresados, color, fuente, visibilidad) y pueden ser vinculados a columnas en tablas de una base de datos. Además pueden ser extendidos de otras subclases y se pueden combinar para crear componentes más complejos. SOFIA provee alrededor de 50 de estos componentes.

[47] Salmon LLC. *SOFIA User Guide*

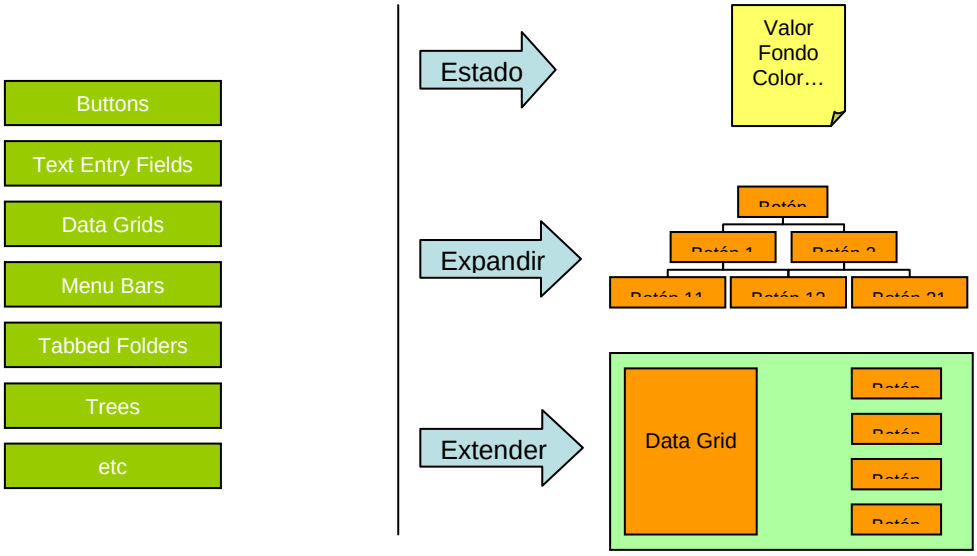


Figura 3.10: Smart GUI Components
Fuente: Salmon Documentation

Observer Pattern: Trabaja como un direccionador de eventos. Si un componente necesita ser notificado de la acción de un usuario o del requerimiento de una página, estos pueden implementar un 'escuchador' de eventos que notifica cuando una actividad en particular sea ha realizado. Este es el mejor modo de manejar los eventos de un usuario. Existen varios eventos dentro de una página:

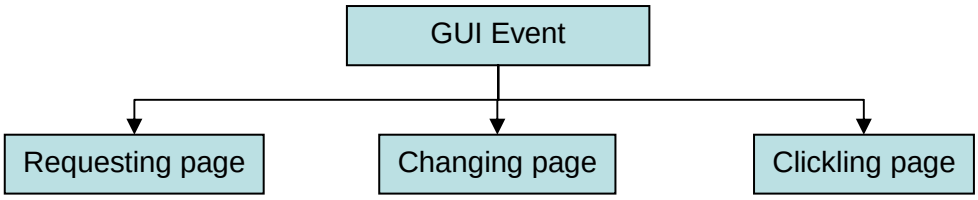


Figura 3.11: Manejo de Eventos
Fuente: Salmon SOFIA Documentation

Datastore Component Abstracts SQL: Mediante esta arquitectura no se tiene escribir sentencias SQL directamente, sino que se interactúa con la base de datos a través de conectores. Las aplicaciones se vuelven independientes de la base de datos.

Session Manager: Toda información que necesita persistir entre el requerimiento de una página y otra es almacenada en la sesión. Esta es una característica muy útil en J2EE. La sesión es un área global de memoria donde cualquier objeto puede ser almacenado bajo un nombre único.

Parallel Inheritance: Este concepto permite extender una página JSP de otra página base que contiene los elementos comunes que se desean mostrar (por ejemplo la cabecera y el pie de página). Junto con esto también es posible tener un controlador base (Base Controller) para manejar todos estos elementos comunes al igual que servicios no visuales que son necesarios para la ejecución de toda la aplicación.

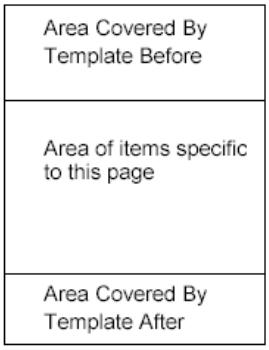


Figura 3.12: Parallel Inheritance
Fuente: Salmon SOFIA User Guide
Autor: Salmon LLC

3.7.3. Elementos y fundamentos de SOFIA

3.7.3.1. Componentes de la vista

Estos componentes se derivan de los *Smart GUI Components*. Dentro del framework existen varios componentes que van en las JSP y se los puede crear de 2 formas:

- A través de código Java para instanciarlos posteriormente dentro de la página. Esto permite tener control sobre los componentes y permite la su creación "al vuelo" en etapa de ejecución, pero puede volver tediosa su creación.

- La otra forma es crearlos mediante el uso de la librería de tags JSP personalizada de SOFIA (*The Salmon Open Framework JSP custom tag library*). Esta librería contiene tags que remplazan y se extienden de los tags HTML primitivos. Estos componentes de SOFIA son similares a los que presenta el HTML nativo.

Existen componentes son tanto visuales (tablas, botones, textos, etc.) como no visuales (datasources, parámetros, internacionalización, etc.). La mayor cantidad son componentes visuales, algunos de los cuales remplazan a los tradicionales HTML, pero existen otros que permiten hacer tareas más complejas. A continuación se listan los componentes visuales más importantes de SOFIA:

General Visual Components	Form Visual Components
Container	Button
Data Table (Multiple Rows)	Check Box
Detail Form	File Upload
Display Box	Hidden Field
Font	Submit Image
Image	Password Edit
Link	Radio Button
List	Radio Button Group
List Form	Drop Down List
Navigation Bar	Submit Button
Raw HTML	Text Area
Table (Single Rows)	Text Edit
Text	Date Entry
Tree	Decimal Number Entry
Cookie Crumble	Email Entry Component
Brand Image Component	Fraction Entry
Calendar	Look Up Button
CSS Style Component	Phone Number Entry
Box	Social Security Number Entry
	US Sates Drop Down List
	Zip Code Entry
	Row Selector
	HTML Form Tag

Cuadro 3.9: Componentes Visuales de la SOFIA
Fuente: Salmon SOFIA Overview Presentation, Salmon LLC Site
Autor: Dan Dubinsky, SOFIA Lead Architect

Los componentes no visuales no tienen una representación visual al ejecutar la aplicación, sino que realizan varias tareas invisibles al usuario (como conexión a la base de datos o validaciones).

Todo componte no visual es representado por un pequeño icono dentro de Dreamweaver. Estos iconos ayudan a localizar estos componentes en la etapa de desarrollo pero no se muestran en el browser cuando se ejecuta la página. Los principales componentes no-visuales son:

Non-Visual Components	Non-Visual Components Cont	Foundation Components
Datastore	Message Logger	HTML Page
Property	Sorting	JSP Controllers
Body	Generate URL	Model
Acrobat Form	Byte Parsing	
Send Email	Holiday Calendar	
IDE and HTLM Tool Integration	Server Performance Monitoring	
Internationalization		
Word Mail Merge		
Property Configuration File		
Task Scheduler		
SecureID Server		

Cuadro 3.10: Componentes No-Visuales de SOFIA
Fuente: Salmon SOFIA Overview Presentation, Salmon LLC Site
Autor: Dan Dubinsky, SOFIA Lead Architect

Todos los componentes nombrados anteriormente son utilizados para la construcción de la vista y tienen perfecta integración con Macromedia Dreamweaver MX a través de la extensión adicional. Una página puede ser construida simplemente arrastrando cada uno de estos componentes en la posición deseada.

3.7.3.2. Archivos de propiedades (Property Files)

Uno de los problemas más comunes cuando se desarrolla grandes aplicaciones es mantener una consistencia con respecto a la interface de usuario. Generalmente se

tiene gran cantidad de páginas hechas por diferentes personas y se necesita que luzcan más o menos igual.

SOFIA utiliza archivos de propiedades exclusivos de cada aplicación. Son usados para describir las características por defecto de presentación de la aplicación. Trabajan de manera similar a las hojas de estilo en cascada (CSS) definiendo colores o fuentes que por defecto nuestra aplicación. Los *Property Files* pueden utilizarse en conjunto o remplazando los CSS's. Adicionalmente estos archivos contienen información de configuración tal como directorios de archivos vitales o rutas de acceso a la base de datos.

Existe un archivo de propiedades principal llamado *System.properties*, este archivo contiene las características por defecto que utilizaran todas y cada unas de las aplicaciones del framework. Adicionalmente las páginas que están agrupadas en cada aplicación pueden utilizar los valores dados por un archivo de propiedades exclusivo independiente denominado *NombreDeAplicacion.properties*. Estos archivos de propiedades trabajan de manera jerárquica, por esto cada una de las configuraciones especificadas serán buscadas primero en el archivo de propiedades de la aplicación y si no son encontrados aquí se los buscara en el archivo de propiedades principal.

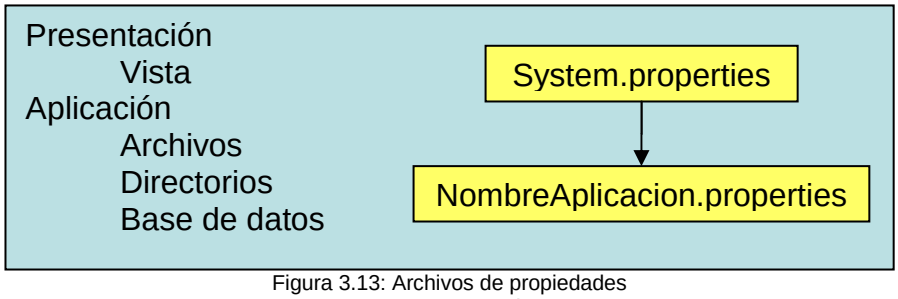


Figura 3.13: Archivos de propiedades
Autor: Edison Hernández

Dentro de las configuraciones que se encuentra en estos archivos se destaca la conexión a base de datos:

```
DBDefaultProfile= NombreAplicacion
NombreAplicacion.DBDriver=
NombreAplicacion.DBURL=
NombreAplicacion.DBUser=
NombreAplicacion.DBPassword=
```

Una aplicación de SOFIA puede utilizar varias bases de datos a la vez, simplemente hay que especificar un nuevo perfil dentro de este archivo y especificar los parámetros de acceso.

Además, contiene características por defecto de vista y configuración de los diferentes componentes de la interface gráfica:

```
PageBackgroundColor=
PageBackground=
PageTextColor=
PageLinkColor=
PageVisitedLinkColor=
PageActiveLinkColor=
PageMarginHeight=
PageMarginWidth=
PageRightMargin=
PageLeftMargin=
PageTopMargin=
```

Cada componente es automáticamente inicializado con las propiedades visuales dadas, pero se pueden sobrescribir estas características mediante el uso de CSS, propiedades HTML dentro de las páginas JSP o en el controlador de la página. Si no

se da valor alguno a un componente, sus características serán dadas por el archivo de propiedades principal (*System.properties*).

También se pueden definir tipos de fuente que va a utilizar la aplicación:

```
DefaultFont.StartTag = <FONT FACE="Verdana" STYLE="FONT-SIZE:14;" COLOR="BLACK">
DefaultFont.EndTag = </FONT>
#
EmphasisFont.StartTag = <FONT FACE="Verdana" STYLE="FONT-SIZE:14;"
COLOR="BLACK"><B>
EmphasisFont.EndTag = </B></FONT>
#
PageHeadingFont.StartTag = <FONT face="Verdana, Arial, Helvetica, sans-serif"
color="#732b0A" size="5"><B>
PageHeadingFont.EndTag = </B></FONT>
```

Hay que remarcar que todos los cambios hay que efectuarlos en el archivo de propiedades de la aplicación (NombreDeAplicacion.properties) y no en el archivo de propiedades principal (System.properties). Esto se debe a que el archivo principal es creado en la instalación de SOFIA y es modificado cada vez que se crea un nuevo proyecto, en cambio el archivo de propiedades de la aplicación es creado cuando se crea el proyecto y no es modificado posteriormente por el framework.

3.8. DESARROLLO DE APLICACIONES EN SOFIA

3.8.1. Manejo de proyectos

Eclipse es el "centro de operaciones" de SOFIA. Mediante éste se crean y se ejecutan los proyectos, además es el editor de código Java entre otras cosas.

Lo primero que se debe hacer antes de iniciar un nuevo proyecto, es asegurarse que la base de datos esté funcionando correctamente. Posteriormente se ejecuta Eclipse, que presenta un menú personalizados de SOFIA si se ha realizado correctamente la instalación.

Figura 3.14: Barra de herramientas de SOFIA en Eclipse
Fuente: Eclipse

Botón	Descripción
1. Iniciar el servidor	Este botón inicia el servidor web para poder ejecutar la aplicación. Además es necesario tener levantado el servidor para trabajar en Dreamweaver.
2. Detener el servidor	Detiene el servidor de aplicaciones.
3. Ejecutar aplicación	Ejecuta la página principal de la aplicación.
4. Ejecutar Dreamweaver	Corre la aplicación Dreamweaver para edición de páginas JSP.
5. Crear Modelo	Crea el modelo de datos.
6. Editar Modelo	Edita un modelo creado previamente.
7. Crear Controlador	Crean un controlador para una página JSP.
8. Crear variables de instancia	Crea las variables que inicializan los componentes de una página JSP en el controlador.
9. Página principal	Define la página principal de la aplicación.
10. Mapa del sitio	Crea el mapa del sitio para manejar nombres lógicos.

Tabla 3.11: Barra de herramientas de SOFIA en Eclipse
Autor: Edison Hernández

Al crear una nueva aplicación en Eclipse, inmediatamente se puede crear la conexión a la base de datos. Para crear un nuevo proyecto se escoge: File/New Project/Java/SOFIA Project/, a continuación aparece una pantalla donde se especifica el nombre del proyecto y variables para conexión a la base de datos:

```
JDBC Jar: \TOMCATHOME\common\lib\mm.mysql-2.0.11-bin.jar
JDBC Driver: org.gjt.mm.mysql.Driver
JDBC URL: jdbc: mysql://localhost/nombreBaseDatos
User ID:
Password:
```

- *JDBC Jar* especifica la ruta al archivo JDBC para la conexión a la base de datos (en el ejemplo se especifica el JDBC de MySQL).
- *JDBC Driver* indica la ruta virtual del JDBC.
- *JDBC URL* indica cual es la base de datos que se va a utilizar
- *User ID* y *Password* son datos de autentificación a la base de datos. Si se va a utilizar el mismo usuario de acceso indicado en la instalación de SOFIA no es necesario especificar estos valores, ya que los coge del archivo de propiedades de principal.

Todos estos datos se crean como variables en el archivo *NombreAplicacion.properties* por tanto estos valores pueden ser modificados posteriormente.

Por defecto todos los proyectos creados están dentro de la carpeta *workspace* en la carpeta de Eclipse aunque se puede especificar otra ubicación. Dentro de la carpeta *workspace* están carpetas por cada una de los proyectos creados.

El momento de crear un proyecto, dentro de cada carpeta se crea una estructura de directorio y archivos como la siguiente:

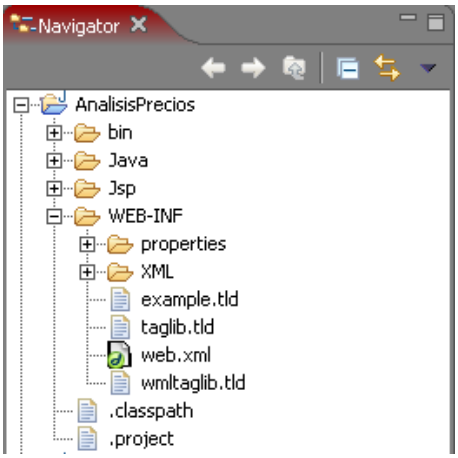


Figura 3.15: Estructura de un proyecto en Eclipse
Fuente: Eclipse

- *bin* contiene los archivos copilados (.class) que resultan de la compilación de una clase Java
- *Java* contiene las clases Java que se utilizarán en el proyecto. Estas clases pueden ser Controladores, Modelos u otras necesarias.
- *Jsp* contiene los documentos de la vista, es decir las páginas JSP
- *WEB-INF* contienes varios archivos y directorios:
    - Directorio *properties* que almacena los archivos de propiedades del proyecto. Este directorio es creado opcionalmente, ya que se puede tener los archivos de propiedades en un directorio común para todos.
    - Archivos de tipo XML son de configuración del proyecto. El archivo que sobresale aquí es el *taglib.tld* que contiene todos los componentes gráficos que pueden ser utilizados dentro de las páginas Jsp, y mediante este archivo Dreamweaver interpreta los tag de cada

componente que contiene el prefijo "*salmon*" dentro de un JSP y los muestra gráficamente.

- *.classpath* contiene rutas a directorios de Tomcat además de paquetes adicionales que se puedan necesitar para la ejecución de la aplicación.
- *.project* contiene una descripción del proyecto.

Para que Dreamweaver pueda traducir los componentes exclusivos de SOFIA dentro de las páginas JSP, el servidor de aplicaciones (Tomcat) se debe estar ejecutando. Esto ocasionalmente puede provocar que los recursos del sistema se disminuyan. A continuación vemos como Dreamweaver interpreta los componentes de SOFIA y los muestra dentro de la vista preliminar de la página:

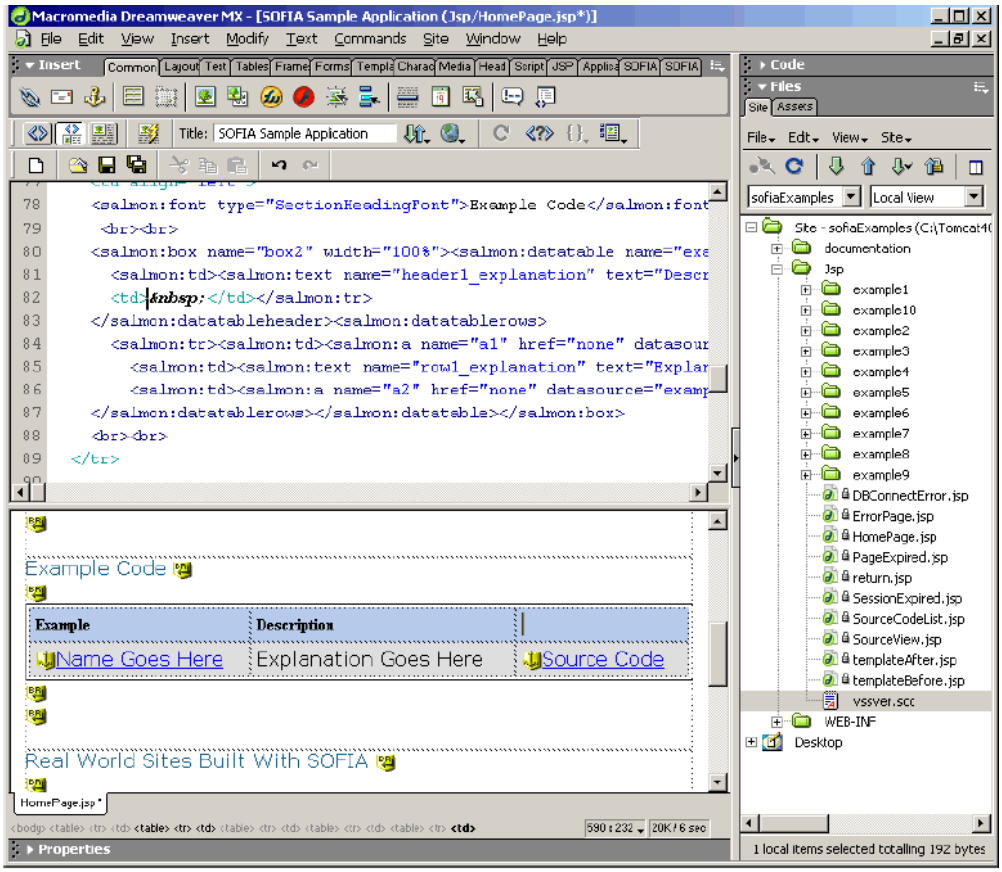


Figura 3.16: Interface de Dreamweaver con SOFIA
Fuente: SOFIA Tutorial Documentation

Mediante esta interface se maneja la estructura de las páginas y componentes web de una aplicación (imágenes, videos, archivos de JavaScript, etc.). Dentro de Dreamweaver también se crean varios menús de herramientas de SOFIA que se detallan a continuación.

3.8.2. Fundamentos de desarrollo

3.8.2.1. La vista

La vista es la implementación de un archivo JSP con tags personalizados de SOFIA. Dentro de Dreamweaver los componentes de SOFIA se pueden manejar de manera fácil, simplemente se los arrastra dentro la página (Drag-and-drop) y aparece una ventana para configurar cada componente.

Existen varios tipos de componentes los cuales están divididos en 3 categorías: Common (Comunes), Input (Ingreso) y Form (Formulario).

La barra de herramientas *Common* contiene los elementos más utilizados dentro de SOFIA; por ejemplo campos de texto o imágenes, pero también componentes más complejos como *datasources* o *datatables*. La mayoría de estos componentes no deben estar necesariamente dentro de un formulario:

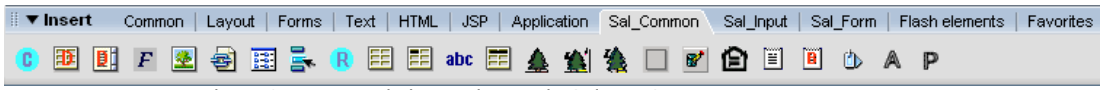


Figura 3.17: Barra de herramientas de Salmon Common en Dreamweaver
Fuente: Aplicación Dreamweaver

La barra de herramientas *Input* contiene elementos que son parte de un formulario (por ejemplo botones, cuadros de texto, check box, listas o cuadros de contraseña). Estos componentes sirven para que el usuario ingrese información que pueda ser procesada de alguna manera. Estos componentes normalmente se los coloca dentro de los tags del formulario:

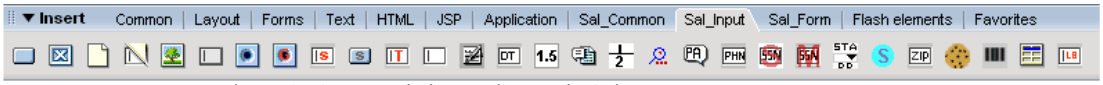


Figura 3.18: Barra de herramientas de Salmon Input en Dreamweaver
Fuente: Aplicación Dreamweaver

La barra de herramientas *Form* es parte de las versiones más reciente de SOFIA. Contiene componentes más complejos (ListForms, SeachForms o DetailForms) además de validadores (Validators) que sirven para controlar errores y mensajes informativos dentro de la página:

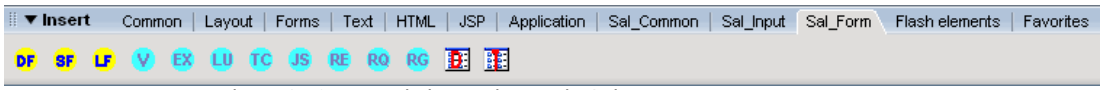


Figura 3.19: Barra de herramientas de Salmon Form en Dreamweaver
Fuente: Aplicación Dreamweaver

Una vez creado el proyecto, el paso siguiente es crear las páginas JSP (la interface gráfica) mediante Dreamweaver. Cuando creamos una nueva página, el framework automáticamente crear el esqueleto del código similar a lo siguiente:

```
01: <%@ taglib uri="/taglib.tld" prefix="salmon"%>
02: <%@ page extends="com.salmonllc.jsp.JspServlet"%>
03: <html>
04: <salmon:page/>
05: <salmon:body/>
06:  <salmon:form name="pageForm">
07:   <salmon:text name="text1" text="Hola Mundo" font="DefaultFont"/>
08:  </salmon:form>
09: <salmon:endbody/>
10: <salmon:endpage/>
11: </html>
```

En el ejemplo anterior se muestra el mensaje "**Hola Mundo**". Este es el punto de partida para construir las páginas JSP.

A continuación se especifica la estructura básica:

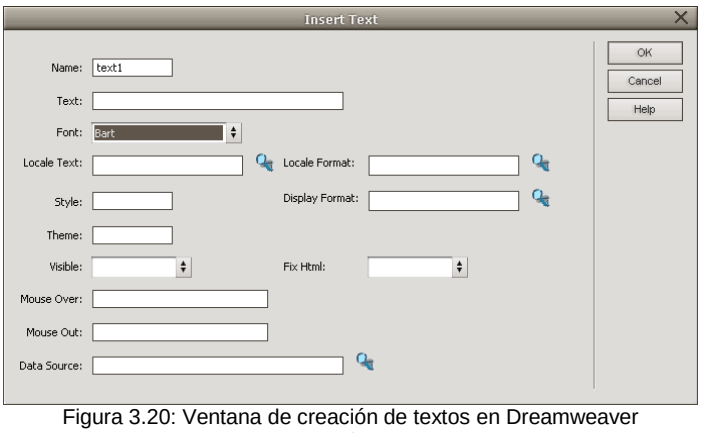
- Línea 01: Específica el directorio de tags JSP. Indica que la página esta usando una librería de *tags* personalizada descrita en un archivo XML llamado *taglib.tld*. Cada tag que contenga el prefijo *salmon* será referenciado a esta librería.
- Línea 02: Esta es la directiva estándar de una página JSP. Indica la clase de la que se extiende la página. Hay que recordar que una página JSP es transformada a una clase Java cuando se ejecutada por primera vez.
- Línea 03 y 11: Delimitan los componentes HTML dentro de una página.
- Línea 04 y 10: Indican el inicio y fin del contenido de la página. Además estos tags sirven para indicar el nombre de la aplicación a la que pertenece la página y los controladores de la página.
- Línea 05 y 09: Delimitan el cuerpo de la página.
- Línea 06 y 08: Indican inicio y fin del formulario. Un formulario es una zona donde se recogen datos ingresados por el usuario para ser posteriormente procesados.
- Línea 07: Contiene un componente de texto de SOFIA

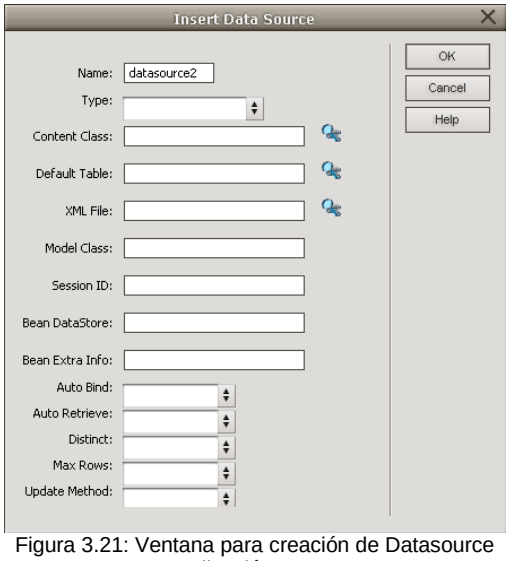
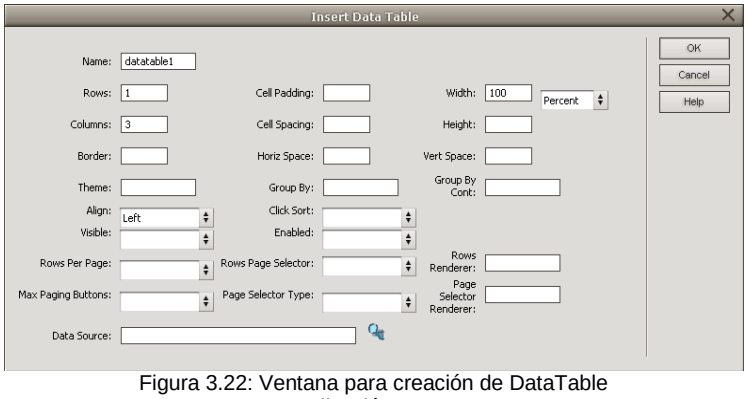
Como se puede apreciar en el ejemplo anterior existen elementos HTML y elementos propios de SOFIA identificados con el prefijo **"salmon"**. La ventaja de los componentes de SOFIA sobre los HTML es que pueden ser manejados y manipulados a través del controlador de la página.

`<salmon:input type="submit" name="submit1" value="Ok">` `</salmon:input>`	Crea un botón que dice *Ok*.

`<salmon:img name="img1" src="images/img1.jpg"/>`	Crea una imagen en la página

A continuación se analizan 3 de los componentes más comúnmente utilizados:

Tipo	Descripción	Atributos
Text	Son textos simples que van dentro de una página JSP. Normalmente se utilizan cuando se quiere mostrar algún tipo de información textual que no pueda ser modificada por el usuario. Su contenido puede ser dado directamente en la página JSP o a través del controlador.  Figura 3.20: Ventana de creación de textos en Dreamweaver Fuente: Aplicación Dreamweaver	Nombre, Contenido, Fuente, Visibilidad, Estilo, Datasource.
DataSource	Permite acceder a información de la base de datos. Un *Datasource* es un componente que indica a la página como obtener la información solicitada. Es un componente no visual que es representado en la vista por un pequeño icono. Puede ser de varios tipos: • *SQL:* Indica que la página se conectará directamente a la base de datos sin intermediarios. Cuando se crea un *Datasource* de tipo SQL, aparece una ventana que permite escoger los campos de la base de datos a los que se quiere hacer referencia. • *Model:* La página llama a la información a través de un modelo de datos (Clase Java que hace referencia a tablas de la base de datos). • *QBE (Query By Example):* Es un tipo especial de *Datasource* que en lugar de obtener los datos, ayuda a construir criterios de selección para llamar a los datos de otro Datasource. Cada elemento en el QBE en lugar de representar una columna en la base de datos, representa una condición de tipo *where*. Es utilizado para hacer búsquedas. Otra característica importante es el *Autoretrieve* que indica cuando se debe llamar a los datos del *Datasource*: • *Never*: Indica que los datos no van a ser cargados nunca. Este es utilizado cuando se quiere controlar como y cuando llamar a los datos (a través del controlador). • *Always*: Los datos son llamados cada vez que la página es cargada. • *OnChange*: Los datos son llamados cuando el criterio de	Nombre, Tipo, Modelo, Autoretrieve.

	selección ha cambiado.  Figura 3.21: Ventana para creación de Datasource Fuente: Aplicación Dreamweaver	
DataTable	Es una rejilla para presentación de registros de la base de datos. Está vinculado directamente a un *Datasource*. Una *Datatable* presenta ordenadamente los registros de un *Datatsource*. Los *Datatable* deben estar dentro un formulario (tag form), ya que tiene botones que se manejan como eventos. Tiene varias funciones incorporadas dentro de su cabecera y su pie: • *La cabecera* contienen los títulos de cada columna y permite el ordenamiento al pulsar sobre estos. • *El pie* contiene botones que indican el número de registros y función de paginación.  Figura 3.22: Ventana para creación de DataTable Fuente: Aplicación Dreamweaver Cuando se vincula el *Datatable* a un *Datasouce* aparece otra ventana que permite escoger los campos a mostrar y su formato.	Nombre, Datasource, Columnas, Filas, Ancho, Alto, Borde, Alineamiento.

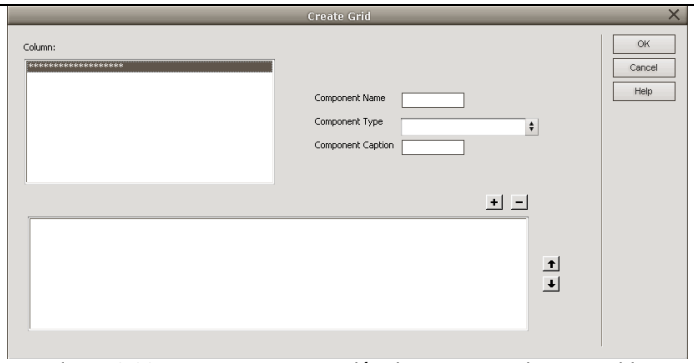


Figura 3.23: Ventana para creación de estructura de DataTable
Fuente: Aplicación Dreamweaver

Tabla 3.12: Componentes más utilizados de de SOFIA
Autor: Edison Hernandez

Con estos 3 simples componentes se puede crear una página compleja que muestre los datos desde una base de datos, mostrarlos en una tabla bien estructurada en donde se pueden hacer ordenamientos y paginaciones. Todo esto se puede hace sin escribir líneas de código.

En una página que por ejemplo muestre los campos *id* y *nombre* de la tabla *usuario* se tiene que crear un *Datasource* que haga referencia a esa tabla, después crear el *Datatable* que haga referencia al *Datasource*. Realizando esto en Dreamweaver se genera el siguiente código:

```
<salmon:form name="pageForm">
<salmon:datatable name="datatable1" model="modelo1" autoretrieve="Always">
   <salmon:datatableheader>
      <salmon:tr>
        <salmon:td>
           <salmon:text name="text1" text="ID" font="TableHeadingFont"/>
        </salmon:td>
        <salmon:td>
           <salmon:text name="text2" text="Nombre" font="TableHeadingFont"/>
        </salmon:td>
      </salmon:tr>
   </salmon:datatableheader>
   <salmon:datatablerows>
      <salmon:tr>
        <salmon:td>
           <salmon:text name="text3" text="ID aquí" font="DefaultFont"
              datasource="datasource1:USER.USER_ID"/>
        </salmon:td>
        <salmon:td>
           <salmon:text name="text4" text="Nombre aquí" font="DefaultFont"
              datasource="datasource1:USER.USER_NAME"/>
        </salmon:td>
      </salmon:tr>
   </salmon:datatablerows>
</salmon:datatable>
</salmon:form>
```

Si se analiza el código anterior generado por Dreamweaver se puede ver que la estructura de los *Datatable* es muy parecida a la de una tabla normal pero está especificado por el tag *<salmon:datatable>* con 2 secciones: *<salmon:datatableheader>* y *<salmon:datatablerows>* que definen la cabecera y el cuerpo de la lista.

Además los textos tienen asignados un datasource de la siguiente manera: *datasource="datasource1:USER.USER_ID"*; esto indica la tabla y columna fuente de datos.

Ejecutando la página se tiene:

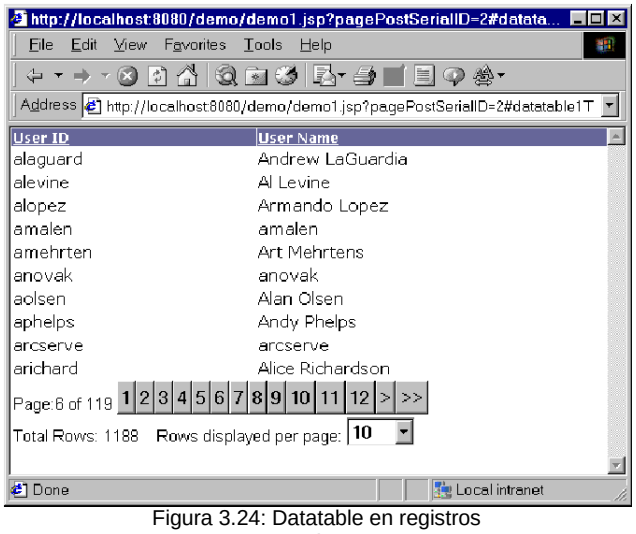


Figura 3.24: Datatable en registros
Fuente: Aplicación Dreamweaver

donde se puede apreciar las secciones de encabezado y pie que permiten el ordenamiento y la paginación de la información.

Los componentes GUI también pueden ser construidos mediante código Java, y más específicamente con los controladores de la página. Para crear un campo de texto editable en el controlador se hace lo siguiente:

```
HtmlTextEdit e = new HtmlTextEdit("edit1", this);
```

y para crear una lista con tres opciones:

```
HtmlDropDownList l = new HtmlDropDownList("list1",this);
l.addOption("1","Option 1");
l.addOption("2","Option 2");
l.addOption("3","Option 3");
```

Si un componente va a ser utilizado varias veces dentro de la aplicación se puede encapsular el componente GUI en una clase Java aparte:

```
import com.salmonllc.html.*;
public class SmartDropDown extends HtmlDropDownList {
        public SmartDropDown(String name, HtmlPage p) {
                super(name, p);
                addOption("1","Option 1");
                addOption("2","Option 2");
                addOption("3","Option 3");

        }
}
```

así entonces solo es necesario instanciar el componente cada vez que se va a utilizar, pero además pueden crear tags personalizados para hacer la llamada a estos componentes directamente en la página JSP y no a través del controlador.

3.8.2.2. El controlador

3.8.2.2.1. Creación

Un controlador esta directamente vinculado a una página JSP y su función principal es manejar los eventos que ocurren dentro de la página. Por ejemplo, si se tiene un botón dentro de un página, al hacer el usuario clic sobre este botón el controlador es llamado y le dice a la página que acción tomar. Un controlador no es más que una clase Java que se extiende de la clase *com.salmonllc.jsp.JspController* .

Un controlador en realidad no debería tener mucho código incluido porque este debería delegar la mayoría o toda la lógica de negocio a los componentes del modelo de datos y la mayoría de la lógica de interface con el usuario a los componentes de la vista; pero esto sería una condición muy difícil de aplicar. Los controladores integran los dos ámbitos de trabajo.

Dentro de la página simplemente se provee el nombre del controlador para que el generador de código (Eclipse) sepa que clase crear y así asociarlos entre sí. Para especificar el controlador se debe agregar el atributo *controller* dentro del tag <*salmon:page*> de la siguiente manera:

```
<salmon:page
        application="NombreAplicacion"
        controller="PaqueteControladores.PaginaController"/>
```

A una página JSP se le puede asignar un segundo controlador agregando el atributo *secondarycontroller*. Este controlador secundario normalmente es utilizado para asignar un controlador base (BaseController) que puede contener funciones y características genéricas para todas las páginas de la aplicación. Por ejemplo pueden manejar la cabecera y el pie de página comunes en todo el sitio, o la autentificación de usuarios:

```
<salmon:page
        application="NombreAplicacion"
        controller=" PaqueteControladores.ControllerPagina "
        secondarycontroller=" PaqueteControladores.BaseController"/>
```

Los controladores pueden estar dentro de paquetes. Los paquetes no son más que carpetas para organizar los diferentes tipos de clases Java.

Una vez que se asigna el controlador a la página, entonces hay que generar el código propio del controlador, esto se puede hacer en Eclipse (aplastando el botón ), o también se puede generar el controlador a través del navegador web agregando el parámetro *generateCode* a la ruta de la página:

```
http://localhost:8080/myApp/Demo.jsp?generateCode=true
```

Si una vez creado el controlador se hicieron cambios, al generar nuevamente el controlador se sobrescriben estos cambios realizados. Para evitar esto es recomendable generar solamente las variables de instancia de cada componente de la página. Esto se lo puede hacer en Eclipse con el botón  o en el browser agregando el parámetro *vars* de la siguiente manera:

```
http://localhost:8080/myApp/Demo.jsp?vars=true
```

3.8.2.2.2. Estructura del controlador

Una vez generado el controlador, se tiene una estructura como la siguiente:

```
package PaqueteControladores;

import com.salmonllc.jsp.*;
import com.salmonllc.html.events.*;

public class MiController extends JspController implements SubmitListener, PageListener{
        public void initialize() {
        }
        public boolean submitPerformed(SubmitEvent event) {
                return true;}
        public void pageRequested(PageEvent event) {
        }
        public void pageRequestEnd(PageEvent event) {
        }
        public void pageSubmitEnd(PageEvent event) {
        }
        public void pageSubmitted(PageEvent event) {
        }
}
```

Al inicio del controlador se especifica el paquete donde se encuentra la clase (*package*) y después se hace llamado a las librerías (*import*). Adicionalmente existen un conjunto de métodos que se detallan a continuación

Variables de instancia

Un controlador obtiene los componentes de la vista de manera general así:

`HashTable getComponentTable();`	`:Lista de Componentes`
`HtmlComponent getComponent(String name);`	`:Clase de un Componente`

Esto se lo debe hacer por cada componente que se va a utilizar en el controlador. Si por ejemplo se tiene en la página JSP lo siguiente:

```
<salmon:form name="pageForm">
   <salmon:text name="txtMiTexto" text="Mi Texto"/>
   <salmon:input type="submit" name="btnMiBoton" value="Mi Boton"></salmon:input>
</salmon:form>
```

Dentro del controlador en las primeras líneas se crean las variables de instancia para cada uno de los componentes de la vista. En el ejemplo se tiene 3 componentes: un botón llamado *btnMiBoton*, un texto *txtMiTexto* y el formulario de la página *pageForm*. Las instancias se crean así:

```
public com.salmonllc.html.HtmlSubmitButton _btnMiBoton;
public com.salmonllc.html.HtmlText _txtMiTexto;
public com.salmonllc.jsp.JspForm _pageForm;
```

Inicialización

Después de la creación de las variables de instancia, cada componente puede ser utilizado dentro del controlador. Existen varias funciones para manejar estos componentes. La función *initialize()*permite inicializar valores o componentes. Es el único método del controlador que es llamado automáticamente por el framework y pasa solo una vez, la primera vez que la página es requerida en la sesión de usuario.

```
public void initialize(){
     _txtMiTexto.setText("Hola Mundo");
}
```

Esto inicializa el texto a "*Hola Mundo*" que aparecerá en txtMiTexto la primera vez que el usuario ejecuta la página. Si no se ha creado la variable de instancia de un componente, se lo puede inicializar de la siguiente manera:

```
public void initialize(){
  com.salmonllc.html.HtmlText t=(com.salmonllc.html.HtmlText)getComponent("txtMiTexto");
  t.setText("Hola Mundo");
}
```

Eventos

La función principal del controlador es manejar los eventos que ocurren dentro de una página. Existen varias clases de eventos entre los que se destacan:

	Descripción	**Parámetros**	**Retorno**
submitPerformed	Este evento es ejecutado por el componente HTML que dispara la página. Recibe como parámetro el componente que causo el evento.	*SubmitEvent event*	*Boolean*: Retorna *verdadero* para continuar procesando eventos o *falso* para detener
pageRequested	Este evento se dispara cada vez que la página es requerida por el browser, y antes de generar cualquier tipo de código HTML	*PageEvent event*	*Void*: No retorna nada
pageRequestEnd	Este evento se dispara cada ve que la página es requerida por el browser después de la generación de código HTML.	*PageEvent event*	*Void*: No retorna nada
pageSubmitEnd	Este evento ocurre cada vez que página es disparada después de que todos los valores son copiados desde el formulario HTML al componente del framework que representa cada campo.	*PageEvent event*	*Void*: No retorna nada
pageSubmitted	Este evento ocurre cada vez que página es disparada, y antes que cualquier valor sea copiado desde el formuladio HTML al componente del framework que representa el campo.	*PageEvent event*	*Void*: No retorna nada
valueChanged	Este evento es disparado por el componente HTML cuyo valor fue modificado. Recibe como parámetro el componente que realizo la petición.	*ValueChangedEvent event*	*Boolean*: Retorna *verdadero* para continuar procesando eventos o *falso* para parar de procesarlos

Tabla 3.13: Eventos de un controlador
Fuente: Sofia User Guide
Autor: Edison Hernández

Ya que el manejo de eventos está implementado en SOFIA, el programador solo tiene indicar la acción cuando el evento es ejecutado. Por ejemplo si tuviéramos una página con un botón y un campo de texto, podríamos decir que cuando se presione el botón, el texto cambie:

```
public com.salmonllc.html.HtmlSubmitButton _btnMiBoton;
public com.salmonllc.html.HtmlText _txtMiTexto;

public void initialize(){
        _btnMiBoton.addSubmitListener(this);
}
public boolean submitPerformed(SubmitEvent event) {
        _txtMiTexto.setText("Adios");
        return true;
}
```

Se puede ver que en la función *initialize()* inicializa el botón mediante el método *addSubmitListener()*. Este código generado automáticamente por el controlador indica que se debe notificar cuando el usuario haya aplastado el botón. Posteriormente la función *submitPerformed()* indica que acción tomar al presionar el botón.

Al existir varios tipos de eventos, estos se ejecutan en secuencia. Esta secuencia esta establecida de antemano y es de mucha importancia conocerla.

Por ejemplo cuando un botón es presionado no solo se ejecuta el método *submitPerformed()* sino que también se ejecutan los métodos *pageRequested()* y *pageRequestEnd()*. Si una página es ejecutado por primera vez entonces el método *initialize()* es llamado primero y posteriormente *pageRequested()*.

El siguiente gráfico especifica la secuencia de eventos:

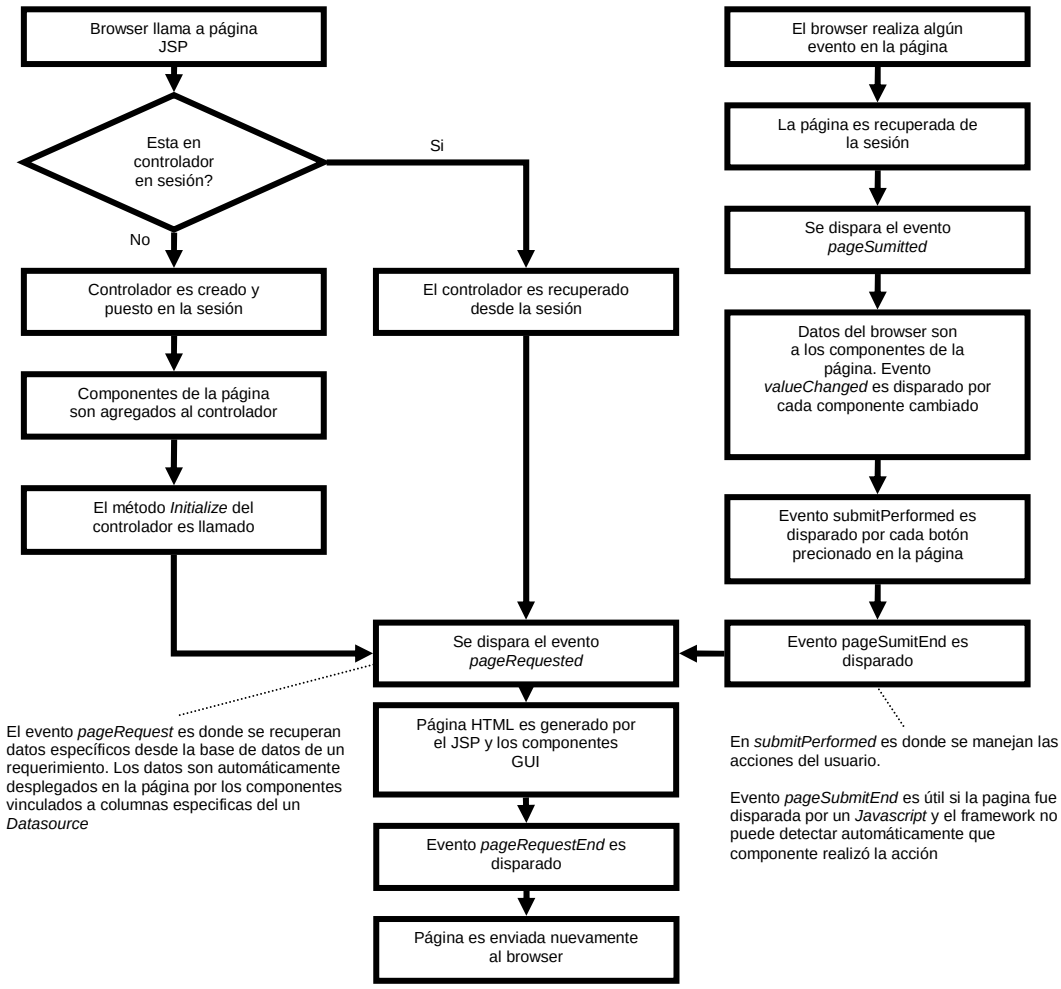


Figura 3.25: Secuencia de Eventos
Fuente: Sofia User Guide
Autor: Salmon LLC

Cuando la página es requerida por el browser, el controlador es llamado. Si el controlador no esta presente en la sesión, entonces es creado y los componentes de la página son agregados a este controlador para después llamar al método *initialize()* y por último al método *pageRequested()*. Si el controlador esta presente en la sesión desde el inicio, entonces simplemente se hace el llamado a *pageRequested()*.

Si al contrario una página es disparada (submit) por un botón, entonces la página es cargada desde la sesión y el evento *pageSubmitted()* es ejecutado. Los datos del formulario de la página son copiados y cargados en memoria y el evento *valueChanged()* es disparado por cada campo que se haya modificado. Posteriormente el evento *submitPerformed()* es disparado por el botón. Adicionalmente a esto, el evento *pageSubmitEnd()* se ejecuta para concluir con el método *pageRequested().*

Una vez que el método *pageRequested()* termina de ejecutarse, la página HTML es generada por el JSP. Finalmente el evento *pageRequestEnd()* se dispara y la página es enviada al browser para que pueda ser visualizada.

3.8.2.3. El modelo

3.8.2.3.1. DataStoreBuffer

El modelo es un componente donde son almacenados los datos para que puedan ser accesados por la vista y el controlador. El objeto *DataStoreBuffer* en el paquete com.salmonllc.sql es la base de todas las clases modelos derivadas.

Un modelo es una representación no-visual de un grid (rejilla) de datos (por ejemplo una hoja electrónica). Los datos pueden ser cargados en la rejilla y referenciados por una combinación de la posición de la fila y la columna. Las filas son referenciadas por un número que empieza desde 0 y las columnas por un número o por su nombre.

Se puede obtener los datos del *DataStoreBuffer* de la siguiente forma:

```
String dato = _dS.getString(0,"col1")
```

donde el primer parámetro del método *getString* es el número de fila y el segundo parámetro es el nombre de la columna.

Igualmente grabar los datos del *DataStoreBuffer* es bastante simple:

```
_dS.setString(0,"col1","nuevo valor")
```

donde los primeros dos parámetros son similares a *getString*. El tercer parámetro es el nuevo valor a ingresar.

Una vez que los datos están en el *DataStoreBuffer*, se pueden manipular de varias maneras. Se puede ordenar, hacer búsquedas, exportar a archivos XML y planos, o crear columnas calculadas. Además, es posible conectarse directamente a la base de datos en caso que se desee ejecutar sentencias SQL directamente.

3.8.2.3.2. DataStore

El objeto *DataStoreBuffer* es un contenedor de datos útil y valioso, pero su utilidad es mayor en sus subclases, ya que estas subclases permiten automáticamente poblar y persistir los datos que contienen.

La subclase más usada es *DataStore* que puede automáticamente generar sentencias SQL para llenar el buffer desde una base de datos relacional y puede también generar las sentencias SQL para grabar los cambios que hace el usuario.

Un *DataStore* puede ser creado en Java en cualquier etapa de la ejecución de una aplicación de la siguiente manera:

```
DataStore dsEjemplo = new DataStore(getApplicationName());
dsEjemplo.addColumn("Tabla1","Columna1",DataStore.DATATYPE_INT);
dsEjemplo.addColumn("Tabla1","Columna1",DataStore.DATATYPE_INT);
dsEjemplo.addColumn("Tabla1","Columna1",DataStore.DATATYPE_INT);
```

donde primero se crea el *DataStore* especificando el nombre de la aplicación de la que va a ser parte mediante la función *getApplicationName()*. Después se debe crear las columnas con el método *addColumn* que recibe como parámetros el nombre de la tabla, la columna y el tipo de dato que se va a almacenar.

Un *DataStore* está vinculado directamente a la base se datos de acuerdo a las tablas y columnas especificadas. Para recuperar los datos de una tabla y almacenarlos en el *Buffer* de un *DataStore* se utiliza el método *retrieve()*:

```
dsEjemplo.retrieve();
dsEjemplo.retrieve("Columna1 = 1000");
```

Este método puede retornar todos los valores que contiene el *DataStore* o filtrarlos simulando una sentencia *where* de SQL. En el ejemplo anterior se filtran los datos con la condición *"Columna1"* igual a*"1000"*.

También se puede filtrar los datos de otras maneras, por ejemplo para recuperar los datos de una juntura (join) de tablas se hace lo siguiente:

```
DataStore ds = new DataStore(getApplicationName());
ds.addColumn("customer.id",DataStore.DATATYPE_INT);
ds.addColumn("customer.name",DataStore.DATATYPE_STRING);
ds.addColumn("account.id",DataStore. DATATYPE_INT);
ds.addColumn("account.balance",DataStore.DATATYPE_FLOAT);
ds.addJoin("customer.account","account.id",false);
ds.retrieve();
```

donde se puede ver como un *DataStore* almacena la información de 2 tablas: *customer* y *account*. Después se saca la información de estas tablas solo si cumple la condición dada (campo *account* de la tabla *customer* igual al campo *id* de la tabla *account*).

La información de los datasources también puede ser acuatizada. Por ejemplo si se tiene:

```
DataStore ds = new DataStore(getApplicationName());

ds.addColumn("customer.id",DataStore.DATATYPE_INT);
ds.addColumn("customer.name",DataStore.DATATYPE_STRING);
ds.addColumn("account.id",DataStore. DATATYPE_INT ,true,true);
ds.addColumn("account.balance",DataStore.DATATYPE_FLOAT,false,true);

ds.addJoin("customer.account","account.id",false);


ds.retrieve("account.balance = 1000");

float balance=ds.getFloat(0,"account.balance");

ds.setFloat(0,"account.balance" , balance + 100.00);
ds.update();
```

se puede ver que se hace la juntura entre *account* de *customer* y *id* de *account*. Después se filtra aun más estos valores a los registros que cumplen la condición "*balance = 1000*". Posteriormente el balance de un registro en particular es incrementado en 100 y finalmente se actualiza el *Datastore* para guardar los datos dentro de la base mediante el método *update()*. Además nótese que cuando se agregan las columnas *account.id* y *account.balance*, se pasan 2 parámetros booleanos adicionales: el primer parámetro indica si la columna es clave primaria y el segundo indica si la columna puede ser modificada o no.

3.8.2.3.3. Modelos reutilizables

Los *DataStore*, como se vio anteriormente, son creados y utilizados dentro de la ejecución, pero una las ventajas de los modelos es que se pueden crear una sola vez y utilizarse varias veces dentro de la aplicación. La manera apropiada de hacer esto es definiendo el *DataStore* como una clase separada:

```
package com.demo;
import com.salmonllc.sql.*;

public class CustomerAccountModel extends DataStore {

        public static final String CUSTOMER_ID = "customer.id";
        public static final String CUSTOMER_NAME = "customer.name";
        public static final String CUSTOMER_ACCOUNT = "customer.account";
        public static final String ACCOUNT_ID = "account.id";
        public static final String ACCOUNT_BALANCE = "account.balance";

        public CustomerAccountModel(String applicationName) {
                super(applicationName);
                addColumn(CUSTOMER_ID,DATATYPE_INT);
                addColumn(CUSTOMER_NAME,DATATYPE_STRING);
                addColumn(ACCOUNT_ID,true,true);
                addColumn(ACCOUNT_BALANCE, DATATYPE_FLOAT,false,true);
                addJoin(CUSTOMER_ACCOUNT,ACCOUNT_ID,false);
        }
}
```

y después crear una instancia del modelo:

```
DataStore ds = new CustomerAccountModel(getApplicationName());
ds.retrieve(ds.CUSTOMER_ID + "=1000");
float balance=ds.getFloat(0,ds.ACCOUNT_BALANCE);
ds.setFloat(0,ds.ACCOUNT_BALANCE, balance + 100.00);
ds.update();
```

Aquí se filtra los datos y se actualiza el datasource, pero la juntura ya no se la hace aquí, sino que es parte del modelo. Por tanto siempre que se llame al modelo se ejecutará la juntura.

Para la creación de modelos no es necesario escribir todo ese código sino que el framework lo genera automáticamente. Para esto en Eclipse se presiona el botón  y aparece una ventana que guía en la creación de los modelos.

3.8.2.3.4. DataSource

Un DataSource sirve para vincular un *DataStore* a la vista de diferentes formas. Si se tiene un modelo reutilizable se lo puede vincular a la vista de la aplicación de la siguiente manera:

```
<salmon:datasource name="dsAccount" type="MODEL" model="com.demo.CustomerAccountModel">
</salmon:datasource>
```

Nótese que el *DataSource* es de tipo *MODEL* lo que le indica que su fuente es un modelo. Cuando se ha creado un *DataSource* en la vista, igual que cualquier otro componente que se haya creado, se lo instancia en el controlador para que pueda ser utilizado posteriormente:

```
CustomerAccountModel ds = (CustomerAccountModel) getDataSource("datasource1");
```

o una alternativa mas común para instanciarlo es así:

```
public com.salmonllc.sql.DataStore _dsAccount;
```

Adicionalmente, si se necesita un *DataSource* que va a ser utilizado en una sola página, entonces se puede crear la referencia a tablas de la base de datos directamente en la página JSP. Aunque esto no se ajusta a los conceptos que maneja el patrón MVC es utilizado para mayor rapidez. El tipo de este *DataSource* es *SQL*:

```
<salmon:datasource name="ds" type="SQL">
<salmon:datasourcedef>
<salmon:field tablename="customer" fieldname="id"/>
<salmon:field tablename="customer" fieldname="name"/>
<salmon:field tablename="customer" fieldname="account"/>
<salmon:field tablename="account" fieldname="id"/>
<salmon:field tablename="account" fieldname="balance"/>
<salmon:join outer="false">
<salmon:joinfield fieldname1="customer.account" fieldname2="account.balance"/>
</salmon:join>
</salmon:datasourcedef>
</salmon:datasource>
```

Este conjunto de tags construyen un *DataStore* similar al que se construyó en los ejemplos anteriores ya que en realidad por debajo lo que hacen estos tags es crear un *DataStore* y llamar a los métodos *addColumn()* y *addJoin()* en la misma forma que el código Java. Estos tags son generados por el framework.

A pesar de la forma en la que se creó el modelo, el acceso y comportamiento es igual, la única diferencia es la forma en la que el modelo es creado e inicializado. Un *DataSource* de tipo SQL es creado e inicializado "al vuelo" por la página JSP. Un modelo en código Java es también creado cuando se requiere, pero la inicialización del código es hecha en el constructor del modelo.

3.8.2.3.5. Vinculo entre DataStores y GUI

Una de las mayores ventajas de un *DataStoreBuffer* es que puede ser vinculado a un componente gráfico de la vista. Una vez que el que el componente esta vinculado a una columna del *DataStoreBuffer*, el valor desplegado en el componente siempre reflejará el valor en la columna del *DataStoreBuffer*, y cualquier cambio hecho por el usuario a este componente será automáticamente copiado al *DataStoreBuffer*.

Para enlazar un componente de la pantalla con un *DataStore* en Java se haría por ejemplo:

```
DataStore ds1 = new DataStore(…);
HtmlTextEdit edit1 = new HtmlTextEdit(…);
edit1.setColumn(ds1,"nombrecolumna");
```

pero también pueden ser vinculados directamente en la página JSP de la siguiente manera:

```
<salmon:input type="text" name="textedit1" datasource="ds1:columnname"></salmon:input>
```

Algo a tomar en cuenta es que el dato que un componente va a mostrar depende del contexto en el que está. Por ejemplo si un campo vinculado a una columna de un *DataSource* es colocado dentro de un contenedor que puede manejar múltiples filas (un *Datatable* o una lista) el componente será repetido en el HTML una vez por cada fila del *DataStore*. En cambio, si el componente es un campo de texto, este solo desplegará una fila del *DataSource*.

```
<salmon:list name="list1" datasource="ds1" >
        <salmon:input type="text" name="textedit1" datasource="ds1:column1"></salmon:input>
</salmon:list>
```

Cualquier tipo de componente puede ser asignado a una columna de un *DataSource*. Cuando se crear un componente, dentro de la ventana de creación se puede vincular el componente con el *DataSource*. La forma en que los datos son mostrados depende del componente: un componente *text* por ejemplo muestra los datos con un tipo de letra en particular, un componente *raw* muestra el texto sin formato, un componente *link* interpreta los datos como el URL de destino, un *input field* muestra los datos y permite la edición de estos.

3.8.3. Componentes avanzados del SOFIA

Dentro del framework SOFIA existen gran cantidad de componentes y utilidades que facilitan y aceleran el desarrollo de aplicaciones, los principales componentes avanzados se detallan a continuación.

3.8.3.1. Contenedores

Los contenedores encapsulan secciones de código dentro de una página, esto permite manejar la lógica de una sección por separado (en diferentes controladores) en lugar de utilizar un controlador para toda la página.

Para crear un contenedor dentro de una página se usa el tag <*salmon:container*>. Este tag tiene como atributo un controlador que hará referencia a los componentes GUI que están dentro del controlador. El controlador del contenedor es una clase que se deriva de la clase *JspContainer*. Este controlador se comporta parecido a los *JspController*, ya que maneja los eventos y la inicialización de componentes de manera similar.

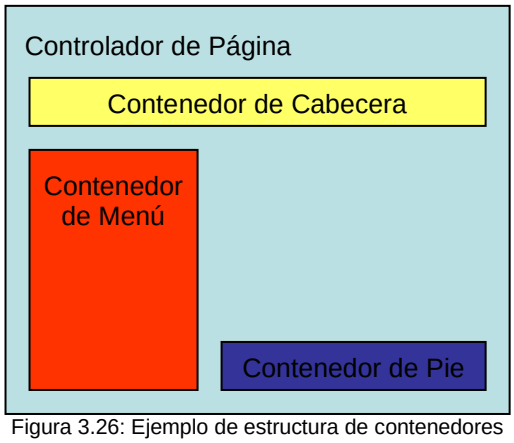


Figura 3.26: Ejemplo de estructura de contenedores
Fuente: Sofia User Guide

Posteriormente el controlador de la página hará referencia a cada uno de los contenedores en lugar de hacer referencia a los componentes GUI por separado. Los controladores de los contenedores manejarán la lógica de los componentes que contienen estos.

```
<%@ taglib uri="/taglib.tld" prefix="salmon"%>
<salmon:container name="container1" containerclass="controllers.ContainerSubClass">
        <salmon:text name="text1" text="Hello" />
        <salmon:text name="text2" text="Hello Again" />
</salmon:container>
```

Los contenedores se los puede crear en páginas JSP separadas y después llamarlos mediante *includes* a la página principal. Esto puede ser útil cuando se tiene secciones que se van a utilizar varias veces en la aplicación (por ejemplos si se tuviera varias páginas que tienen la misma cabecera, el mismo menú y el mismo pie de página). Si existen secciones tienen el mismo comportamiento siempre, entonces se puede agruparlos en contenedores y manejar la lógica de cada uno de ellos independientemente de la página en la que se encuentren.

3.8.3.2. Buckets

Un DataSource puede ser asignado a columnas de una base de datos, pero mediante los *Buckets* se pueden crear columnas que simplemente existen en memoria.

Los *Buckets* funcionan como columnas normales de un *DataSource* pero sus datos se almacenan en un espacio de memoria. Pueden ser vinculados a componentes GUI, filtrados y ordenados como cualquier otra columna. Un *Bucket* a diferencia de una columna no genera sentencias SQL y es creado con el método *addBucket*:

```
DataStore ds1 = new DataStore(getApplicationName());
ds1.addBucket("NombreBucket",DataStore.DATATYPE_INT);
```

donde el primer parámetro es el nombre del *Bucket* y el segundo es el tipo de dato.

3.8.3.3. QBEBuilder

Un *QBEBuilder* es un tipo especial de *DataSource* que es usado para realizar búsquedas en otro *DataSource*. Un *QBEBuilder* no está ligado a columnas de la base de datos, sino a otros *DataSources*. Un *QBEBuilder* puede tener varios elementos, cada elemento es un criterio que representa una porción del filtrado completo.

Estos elementos de selección pueden ser vinculados a GUI para permitir al usuario hacer búsquedas. Internamente esto genera un sentencia *where* de SQL que devuelve los datos del otro *DataSource*. La creación de un *QBEBuilder* se la puede definir en la página JSP así:

```
<salmon:datasource name="datasourceQBE" type="QBE">
  <salmon:qbecriteria name="allColumns" type="complex" columns="*"/>
</salmon:datasource>
```

donde primero se crea el DataSource de tipo *QBE* y posteriormente se crean los criterios de selección deseados. También se los puede definir en código Java creando una clase que se extienda de la clase *com.salmonllc.sql.QBEBuilder*. La creación del *QBEBuilder* se la hace a través de Dreamweaver o Eclipse.

Existen varios tipos de criterios de selección, el más utilizado es el *complex* que permite crear búsquedas complejas donde se puede ingresar varias palabras separadas por el signo '+' para el operador *AND* o el signo '-' para el operador *NOT*.

3.8.3.4. Property Expressions

Un *Property Expression* sirve para indicar a un componente GUI asociado a un *DataSource* como comportarse de acuerdo a los datos que muestra. Por ejemplo permite valuar el contenido de un *textfield* y cambiar su tipo de letra de acuerdo a una condición dada; o si se tiene una imagen cuya ruta la obtiene de una columna en un datasource:

```
<salmon:img name="img1" src="none" datasource="ds1:image_url"/>
```

se podría evaluar la expresión mediante un *Property Expression* para que solo muestre la imagen sin la ruta es diferente de *null*:

```
<salmon:property name="property1" component="img1" propertyname="visible"
datasource="ds1" expression="image_url NOT_EQUAL null"/>
```

lo que esconde la imagen si no existe el atributo *image_url* en el tag *img*.

Estas expresiones son bastante útiles cuando existen componentes que se encuentran dentro de contenedores de datos tales como listas o *DataTables* en donde se muestran múltiples registros. En este caso la expresión es evaluada por cada registro del *DataStore* lo que permite presentar cada registro de manera diferente dependiendo del dato que muestre.

Un *Property Expresión* también puede declararse en el controlador de la página mediante el método *JspController.addPropertyExpression*. Para expresiones más complejas se puede crear una clase Java que se implemente a partir de la clase *DataStoreExpression*.

3.8.3.5. Form Validations

SOFIA ofrece componentes para realizar validaciones de datos. Las validaciones pueden ser hechas tanto en la vista, en el controlador o en el modelo:

3.8.3.5.1. Validaciones en el modelo

La forma más adecuada de hacer validaciones es a través del modelo, ya que de esta forma las validaciones serán llamadas cada vez que el modelo sea utilizado (en 1 o varias páginas). Por ejemplo en un modelo se podría crear una función de validación para requerir que los campos nombre y apellido de un formulario sean obligatorios:

```
private void validateRow(int row) throws DataStoreException {
        if (getContactFirstName() == null || getContactFirstName().trim().length() == 0)
         throw new DataStoreException("Nombre es obligatorio",row,CONTACT_FIRST_NAME);
        if (getContactLastName(row) == null || getContactLastName(row).trim().length() == 0)
         throw new DataStoreException("Apellido es obligatorio ",row,CONTACT_LAST_NAME);
        }
```

y hacer el llamado a esta función cada vez que se vaya a actualizar el *DataStore* sobrescribiendo el método *update*:

```
public void update(DBConnection conn, boolean handleTrans)
throws DataStoreException, SQLException {
        for (int i = 0; i < getRowCount(); i++) {
                if (getRowStatus() == STATUS_MODIFIED)
                validateRow(i);
        }
        super.update(conn, handleTrans);
}
```

lo que provoca que solo se graben los cambios si se ha cumplido la validación de nombre y apellido.

Pero existe otro tipo de forma de validar datos de un modelo. Esto es mediante validaciones declarativas donde no hay que sobrescribir el método *update*, ya que las validaciones son llamadas cada vez que el modelo es utilizado. Las validaciones declarativas son de tipo booleano donde retornan *true* si esta correcto. Hay varios tipos de validaciones declarativas: expresión, tipo de dato, valor requerido, rango requerido, o reglas de JavaScript. Existen funciones específicas para crear cada tipo de validaciones, por ejemplo en:

```
addRequiredRule(CONTACT_FIRST_NAME, "Nombre es obligatorio");
addRequiredRule(CONTACT_LAST_NAME, "Apellido es obligatorio");
setAutoValidate(true);
```

se agregan dos reglas de validación que obligan a ingresar la columnas de nombre y apellido.

A continuación se especifican los tipos de validaciones declarativas que existen:

Tipo	Descripción
Type Check Rules	Sirve para verificar que el valor ingresado corresponde al tipo de dato correcto de la columna del DataStore a la que esta vinculada. Esto es bastante útil para tipos se datos numéricos o de fecha. Si el usuario ingresa un valor diferente que no corresponde al tipo de dato de la columna entonces mostrará el mensaje de error
Expression Rules	Sirve para agregar validaciones más complejas donde se puede por ejemplo crear una sentencia *if* y retorna un valor *true* si es correcto o *false* si es falso.
Lookup Rules	Sirve para verificar si el valor ingresado existe como clave primaria en otra tabla. Es útil cuando se maneja relaciones con claves foráneas y se quiere saber si el valor especificado existe en una o varias tablas.
JavaScript rules	Agrega una validación de JavaScript. Este tipo de validaciones es parecido a *Expression Rules* pero permite crear la sentencia en sintaxis JavaScript. Estas validaciones se ejecutan en el browser y no en el servidor. Retorna un valor booleano.
Required Rules	Sirve para indicar que una columna no puede ser nula y que debe contener un valor.
Regular Expression Rules	Sirve para especificar una expresión regular. El valor ingresado tiene que cumplir la regla especificada para ser valido.
Range Rules	Agrega una regla para indicar el rango de un dato en particular. Se especifica el límite superior e inferior. Si el mínimo no es especificado, solo se evalúa el límite superior de la regla. Si el máximo no es especificado, solo se evalúa el límite inferior de la regla.

Tabla 3.14: Tipos de validadores
Fuente: Sofia User Guide

Estas reglas de validación pueden también especificarse en las páginas JSP mediante el tag *validador*. Cuando se especifican varias reglas a la vez, estas serán procesadas en secuencia y el procesamiento se detiene en la primera regla que fue violada.

Algunas de los tipos de reglas pueden ejecutarse como JavaScript en el browser en lugar esperar que la página sea evaluada cuando los datos se envíen al servidor. Los tipos de reglas que pueden ejecutarse en JavaScript son: Range Rules, Regular Expression Rules y Required Rules.

3.8.3.5.2. Validaciones en Java

Cuando se necesita hacer validaciones más complejas se puede crear una clase que implemente *DataStoreExpression* y que tenga la siguiente estructura:

```
public class myRule implements DataStoreExpression {
        public Object evaluateExpression(DataStoreBuffer dsBuf, int row)
                throws DataStoreException {
                        //REGLA
                }
        }
```

donde la regla es ingresada y cuando se la analiza debe retornar *true* si la validación es correcta o *false* si no lo es. Dentro del modelo se puede llamar a esta validación de la siguiente manera:

```
addExpressionRule(new myRule(), "error message", _componenteParaFocus);
```

Validaciones de este tipo pueden ser útiles por ejemplo si se quiere validar una dirección electrónica en donde se necesita cumplir algunas reglas.

3.8.3.5.3. Mensajes de error

El framework ofrece un componente llamado *HtmlValidatorText* para mostrar mensajes de una validación o mensajes informativos. Este componente es similar a un *HtmlText* con la diferencia que el componente solo será visible si existe un error. Este componente puede ser puesto en la página así:

```
<salmon:validator name="errorMessage" autoimportrules="true" datasource="ds1"
submitcomponents="save" focuslinks="true">
</salmon:validator>
```

y presenta atributos muy útiles que permiten vincularlo a un datasource. Además se puede indicar que ejecute la validación al aplastar un determinado botón. Los validadores posicionan el cursor en el campo donde se produjo el error, entre otras

cosas. El ejemplo anterior valida los datos en el datasource ds1 cuando el botón *save* es presionado.

Para mostrar un mensaje de error en un *validator* se utiliza el método *addErrorMessage* y como parámetros el mensaje de error y el componente que produjo el error:

```
_errorMessage.addErrorMessage("Mensaje de Error",_HtmlComponent);
```

Adicionalmente esto validadores puede desplegar mensajes de error usando mensajes de alerta de tipo JavaScript.

3.8.3.6. Internacionalización

Esta funcionalidad de SOFIA permite controlar el comportamiento y la apariencia de una página web dependiendo del lugar y preferencia de lenguaje del usuario. Para esto se debe especificar atributos correspondientes a *localization keys* en los tags de la interface gráfica. Existen varios componentes que soportan internacionalización:

Componente	Atributo de localización	Atributo afectado
HtmlSubmitButton	DisplayNameLocaleKey	Texto en el botón
HtmlSubmitImage	TextLocaleKey	Texto en el botón para imágenes generadas dinámicamente
	URLLocalKey	El URL de la imagen a desplegar
HtmlTextEdit	DisplayFormatLocaleKey	El formato para ingresar y desplegar valores numéricos o de fecha
HtmlImage	SourceLocaleKey	El URL de la imagen a desplegar
	AltLocaleKey	Texto alternativo que se despliega junto con la imagen
HtmlText	TextLocaleKey	El texto que se despliega
	DisplayFormatLocaleKey	El formato para desplegar valores numéricos o de fecha
HtmlValidatorText	ErrorMessageLocaleKey	El mensaje a desplegar cuando el usuario ingresa un valor que viola una regla
HtmlCalendar		Los formatos de fecha cuando se despliega un calendario.

Tabla 3.15: Componentes que soportan internacionalización
Fuente: Sofia User Guide

El comportamiento por defecto de la internacionalización hace que las diferentes variables para su uso sean cargadas en memoria la primera vez que el lenguaje es requerido por un usuario y permanece en memoria hasta que el servidor es reiniciado.

Para implementar los atributos de localización hay que hacer uso de la clase *com.salmonllc.localizer.Localizer* que se especifica en el atributo *LocalizerClass* del archivo de propiedades de la aplicación. En el directorio de propiedades además se deben crear los *archivos de propiedades de lenguaje* que deben ser nombrados con este formato: *NombreAplicacion.lang.properties* donde *lang* es el lenguaje (*en* para ingles, *es* para español, etc.).

Si este archivo no es encontrado se tratará de cargar el archivo *lang.properties* utilizado por todas las aplicaciones de SOFIA, si este archivo tampoco es encontrado entonces los componentes GUI muestran sus valores por defecto.

Por ejemplo si se quiere aplicar la internacionalización a un texto se lo crea así:

```
<salmon:text name="text1" font="DefaultFont" text="Hello"
textlocalizationkey=”text.hello”/>
```

y en el archivo de lenguajes para español se indica lo siguiente:

```
text.hello=Hola
```

Además la internacionalización se le puede aplicar a través de tablas en la base de datos.

3.8.3.7. Skins

Los *skins* sirven para definir la apariencia de una aplicación. Un *skin* es una colección de atributos donde se detalla el nombre y el valor. Trabajan de manera parecida a los atributos que se declaran en los archivos de propiedades (*properties*). Cuando una propiedad de un componente es indicada en un *skin* y este *skin* es aplicado a una página, la propiedad en el *skin* sobrescribe a la propiedad en el archivo de propiedades.

Dentro de un *skin* las propiedades pueden aplicarse a componentes con nombres específicos o a todos los componentes de un tipo específico.

En el primer caso se denomina **atributos de instancia** que permite aplicar una propiedad a un componente con un nombre en particular de la siguiente forma:

```
instance.componenteNombre.propertyName=value
```

donde se indica el nombre del componente, la propiedad a modificar y el valor. Por ejemplo si se tiene botones con el nombre *btnNuevo* se puede especificar que todos los componentes con este nombre tengan un color específico. Además se pueden aplicar propiedades a grupos de componentes que cumplen una condición en lugar de hacerlo a componentes individuales. Por ejemplo se pueden aplicar una propiedad a todos los componentes que comiencen con un nombre dado, de la siguiente forma:

```
instance.detailField%.visible=false
```

lo que indica que todos los componentes cuyo nombre comiencen con la palabra *detailField* serán invisibles.

El segundo caso para aplicar propiedades a componentes se denomina **atributo de clase** donde pueden aplicar atributos a clases completas de componentes. Esto es similar al primer caso excepto que el atributo de clase se aplica a todas las instancias de una clase donde el *skin* es aplicado.

Los atributos de un *skin* pueden ser de tipo string, entero, boolean o una fuente. Cada componente al que se le va a aplicar un atributo debe tener implementado un método *set* para la propiedad especificada.

Los *skins* pueden residir en varios lugares, el lugar por defecto es el directorio de propiedades de la aplicación, y por convención usa como nombre el siguiente formato: *NombreSkin.skin*. Si se desea también se puede tener almacenado los atributos y propiedades de un *skin* en una tabla de la base de datos.

Una vez que se tiene definido un *skin*, este puede ser aplicado a una página o una aplicación entera. Si se quiere aplicar el *skin* en la programación se pueden utilizar los métodos *setDefaultSkin* para definir el *skin* a toda la aplicación, o *setSkin* para definirlo a una sola página. Aplicar el *skin* en programación permite cambiarlo en etapa de ejecución. Igualmente se puede definir el *skin* para una aplicación en el archivo de propiedades con solo definir el siguiente atributo:

```
SkinDefault=NameOfDefaultSkin
```

3.8.3.8. Site Map

Un mapa del sitio (Site Map) permite crear archivos XML que contienen el mapeo lógico de las páginas JSP, directorios u otros recursos de la aplicación. Cuando se hace referencia a estos recursos dentro de las páginas o controladores, se puede usar el nombre lógico de un recurso en lugar del nombre real. Por tanto si se cambia el nombre de un recurso (por ejemplo de una página JSP), no se tiene que cambiar en todos los lugares donde se hace referencia, solo en el mapa del sitio. Además permite especificar otras propiedades de las páginas (seguridad, ventana pop-up, etc.).

El archivo que contiene el mapa del sitio se encuentra dentro del directorio de propiedades. A cada aplicación se le puede asignar un mapa cuyo nombre sería *NombreAplicaion.map*. Este archivo de mapa es generado por el framework y su estructura es similar a lo siguiente:

```
<entry>

        <name>LogicalName</name>

        <uri>/Jsp/MyJsp.jsp</uri>

        <useForward>false</useForward>

        <secure>false</secure>

        <popupFeatures>default</popupFeatures>

        <action>
                <actionName>OK</actionName>
                <actionEntry>OtherEnrty</actionEntry>
        </action>

        <action>
                <actionName>Cancel</actionName>
                <actionEntry>OtherEntry2</actionEntry>
        </action>

</entry>
```

Los elementos de la estructura del mapa se describen a continuación:

Nombre de elemento	Descripción
name	Nombre lógico para referenciarlo desde otras partes de la aplicación.
uri	Especifica la ruta física del archivo
useForward	Valor booleano que indica si la página usará redirección o no.
secure	Indica si la página debe usar HTTPS o http.
popupFeatures	Es un parámetro opcional que indica si la página deberá abrirse en una ventana pop-up o en una ventana normal. Se puede especificar los parámetros para abrir la ventana.
actions	Sirve para manejar acciones de usuario. Por ejemplo se puede especificar la acción que debe seguir un botón dentro de la página.

Tabla 3.16: Elementos de una entrada en un mapa del sitio
Fuente: Sofia User Guide

3.8.3.9. Message Loggin

Mediante esto se puede tener un registro de mensajes del sistema tanto en consola como en un archivo plano. Hay 5 clases de mensajes que puede ser registrados: aserciones, mensajes de depuración, mensajes de error, mensajes informativos y sentencias SQL. A cada tipo de mensaje se le puede dar un valor de importancia para que muestre únicamente los mensajes de mayor relevancia y evitar que los archivos de registro crezcan demasiado rápido.

Todas las configuraciones para *logs* se las hace en el archivo de propiedades principal (*System.properties*). Aquí se debe especificar el nombre del archivo de *log* (*LogFileName*) y una extensión (*LogFileExt*). Además permite crear un archivo de *log* por cada día (*LogFileAppendDate*).

Por ejemplo si se tiene en el archivo de propiedades lo siguiente:

```
LogFileName=c:\\logs\\log
LogFileExt=dat
LogFileAppendDate=true
```

esto quiere decir que se almacenará así:

"c:\logs\log" + todaysDate + ".dat".

Por defecto cada mensaje indica la hora que ocurrió y el identificador de la amenaza que generó el evento, pero este formato puede ser modificado.

3.8.3.10. List, Detail y List Form

Estos tres componentes tienen por finalidad la creación formularios complejos para entrada y manipulación de datos; y sin tener que escribir líneas de código, ya que encapsulan la lógica internamente para que respondan a eventos y acciones. Por esto no necesita de un controlador para que funcionen. Son útiles cuando se quiere manejar pantallas para mantenimiento de tablas de la base de datos. Estos componentes pueden trabajar en conjunto y permiten hacer búsquedas, listar datos y editar registros, todo esto en un ambiente integrado. Puede trabajar de varias formas:

- La lista y el detalle en páginas separadas: El formulario de búsqueda y el de listado están en una página mientras que el detalle se muestra en otra página.
- Lista y detalle en la misma página: los formularios de búsqueda, listado y detalle están en la misma página.
- Edición en lista: La edición de los datos es hecha en un grid de datos en la misma página de la búsqueda.

La desventaja de estos componentes es que no son muy flexibles, ya que vienen con una estructura predeterminada que no puede ser modificada.

Estos componentes son construidos a partir de otros elementos básicos de SOFIA como validadores (Validators), modelos, QBE y otros.

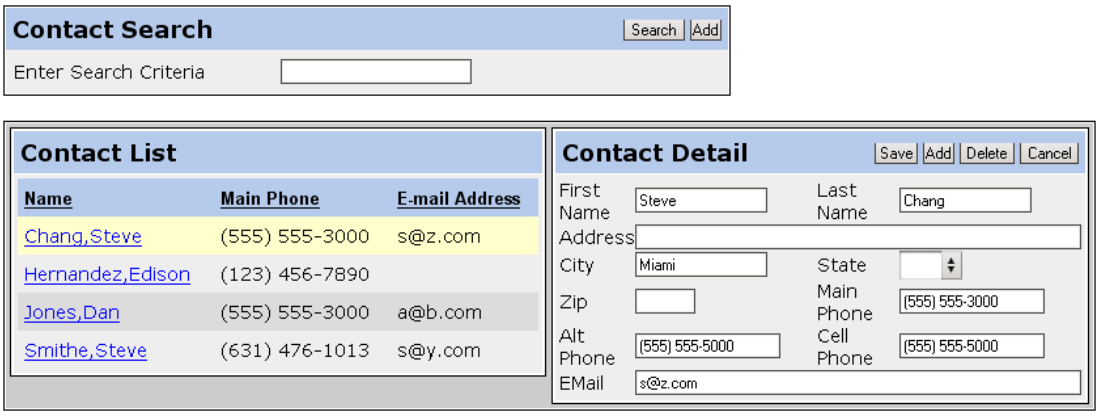


Figura 3.27: Formularios de búsqueda, listado y detalle.
Fuente: Sofia User Guide
Autor: Salmon LLC

Para crear estos formularios es necesario que en la página exista un *DataSource* para que recupere los datos y un *QBEBuilder* para realizar la búsqueda. A partir de esto la creación de los formularios es hecha mediante ventanas de ayuda. Cada uno de los componentes debe ser vinculado a un *DataSource*. Algo a tomar en cuenta es que estos formularios tienen implementado definición de columnas y validaciones básicas, pero se pueden sobrescribir y crear nuevas validaciones en el modelo.

Después de crear los *Datasource* se debe crear el formulario de búsqueda donde se asigna como Datasource el *QBEBuilder*:

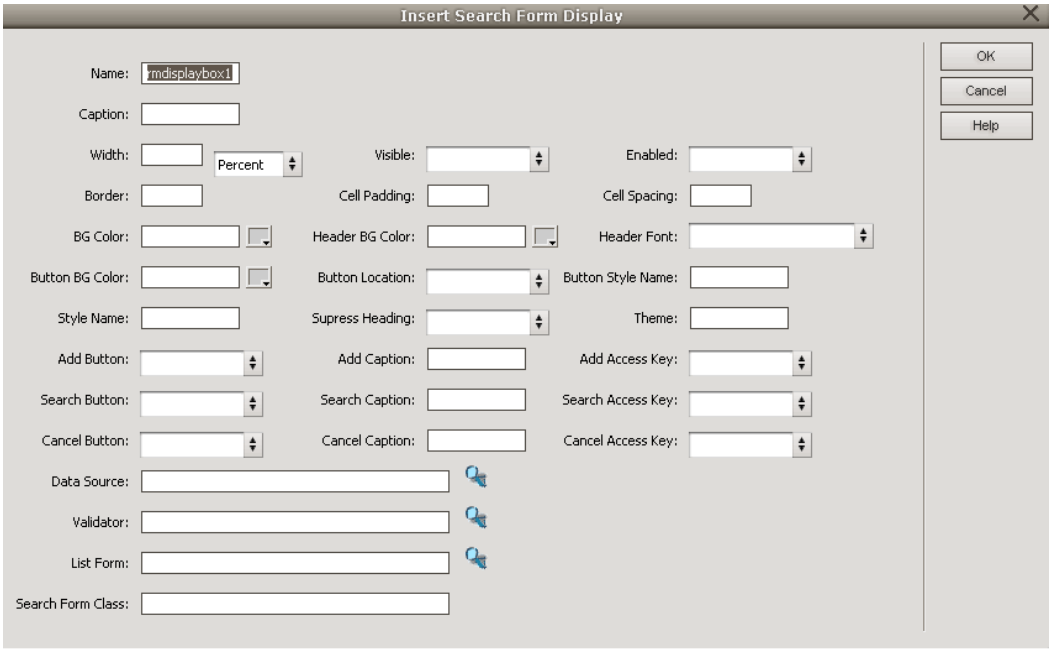


Figura 3.28: Creación de Search Form
Fuente: Sofia User Guide
Autor: Salmon LLC

Después se crea el formulario de listado donde se especifica como *Datasource* el modelo

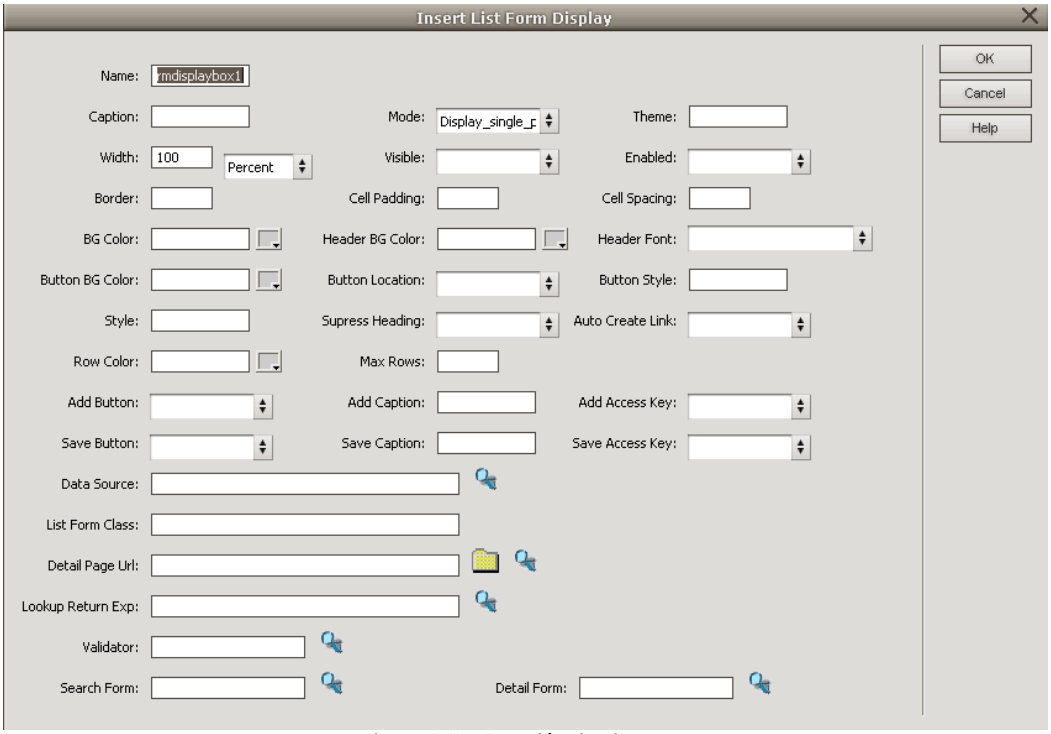


Figura 3.29: Creación de List Form
Fuente: Sofia User Guide
Autor: Salmon LLC

y se crear el formulario de detalle especificando el mismo *Datasource* de la lista

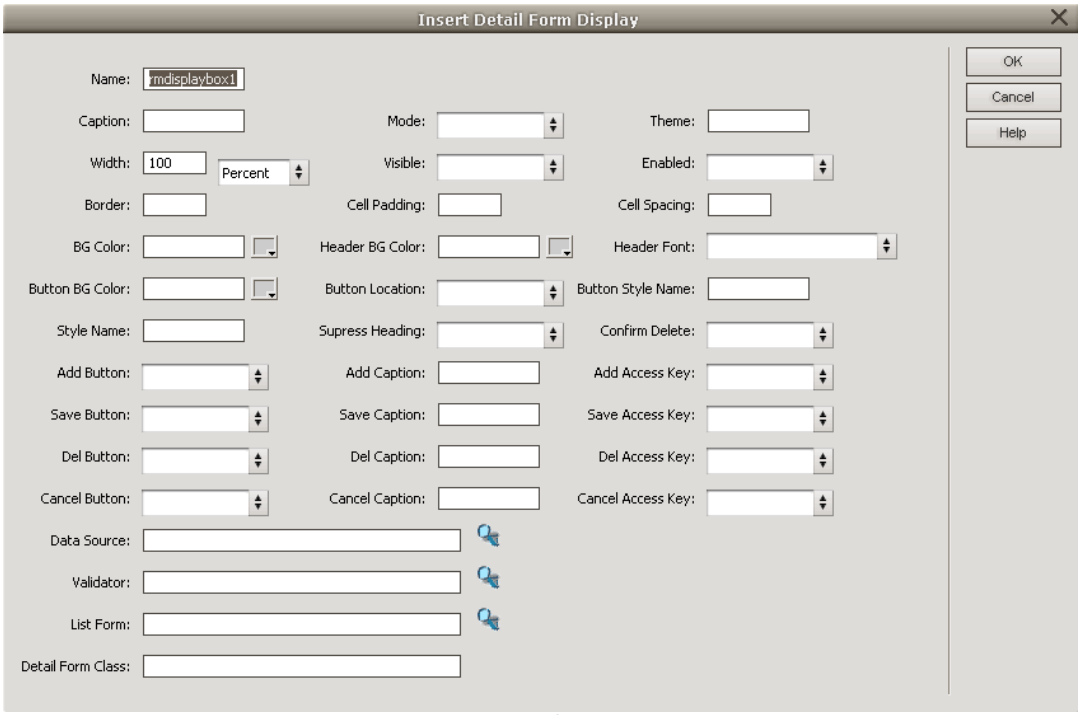


Figura 3.30: Creación de DetailForm
Fuente: Sofia User Guide
Autor: Salmon LLC

Como se puede ver todo esto está dentro de un ambiente integrado donde es muy fácil de trabajar ya que no se necesita escribir código.

Estos componentes trabajan en conjunto por lo que el formulario de búsqueda automáticamente vincula el formulario de listado para mostrar la información. La primera columna de la lista automáticamente se crea como vínculo para que se conecte al formulario de detalle. Los botones para cada uno de los formularios pueden ser mostrados a voluntad.

Cuando los formularios de listado y detalle están en la misma página, el framework automáticamente implementa la conexión entre ellos y permite que se resalte el registro en la lista cuando se muestra en el detalle.

Además si dentro de la página existe algún validador, estos pueden ser vinculados con los formularios para validar de los datos. Finalmente controla que componentes modificados se graben antes de realizar cualquier otra operación, o confirma que un registro se va a borrar.

3.8.3.11. Calendar

Es un componente previamente construido que permite la selección de la fecha mediante una interface estándar. El calendario maneja funciones de JavaScript que permiten seleccionar el mes y el día de manera fácil. Además cuando es ejecutado, automáticamente se muestra el mes y día actual.

Figura 3.31: Calendario
Fuente: Sofia Examples
Autor: Salmon LLC

Viene en 2 tamaños, el más pequeño es similar el de la figura 3.31, pero se lo puede mostrar en un tamaño más grande. Este componente puede ser manejado desde el controlador para marcar una fecha, obtener la fecha que selecciona el usuario o cambiar el tamaño del calendario.

3.8.3.12. Trees

Es un objeto del framework que permite crear árboles de datos. Tiene varias funcionalidades implementadas, por ejemplo permite desplegar o esconder subcategorías, manejar URL para las opciones y mostrar diferentes gráficos dependiendo el nivel jerárquico. Además se puede cambiar colores a través de atributos en los tags o a través de los archivos de propiedades.

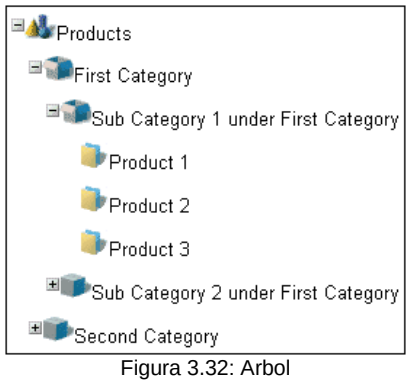


Figura 3.32: Arbol
Fuente: Sofia Examples
Autor: Salmon LLC

Las categorías y opciones del árbol pueden ser construidas en la vista pero también permite cargar las opciones en el controlador o a través de los datos de un modelo si se quiere mostrar valores de una base de datos.

3.8.3.13. Navigation Bar

El framework viene un objeto que permite crear barras de navegación. Estas barras son muy útiles para crear menús dentro de una aplicación. Tiene implementada funcionalidad básica dentro del componente permitiendo cambiar de color la opción seleccionada o esconder un submenú después retirar el puntero. Una barra de navegación puede ser mostrada vertical (Figura 3.33) u horizontal.

Figura 3.33: Barra de navegación
Fuente: Sofia Examples
Autor: Salmon LLC

La construcción se la puede hacer en la página JSP donde se especifica el menú principal y los submenús de cada una de las opciones; pero al igual que los árboles, pueden ser construidos dentro de un controlador o cargar las opciones desde una base de datos. Sus colores y características pueden ser cambiados a través de propiedades en los tags o mediante los archivos de propiedades.

3.8.3.14. Lookup Component

Este tipo de componente sirve para recuperar información de otras páginas desplegando una ventana pop-up. El componente muestra un capo de texto editable y junto a este una imagen que al presionarla despliega una ventana flotante en la que se puede escoger información para que la muestre en el campo de texto.

Este componente es normalmente utilizado para escoger fecha desde un calendario o también para seleccionar información de diferentes tablas de la base de datos. Este componente despliega en la ventana flotante una página nueva que debe ser creada exclusivamente para este fin.

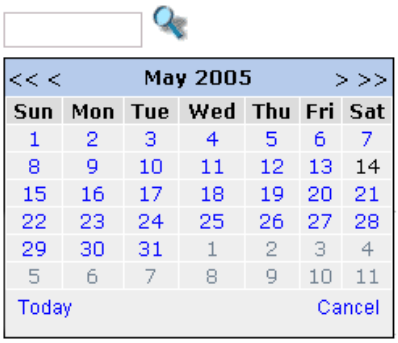


Figura 3.34: Lookup Component
Fuente: Sofia Examples

3.8.3.15. Scheduler

El *Scheduler* es un servlet que permite la ejecución de tareas en intervalos definidos. Para que funcione correctamente debe ser configurado en el archivo *web.xml*. El *Scheduler* puede realizar las siguientes tareas:

- Limpiar conexiones en desuso en el pool de conexiones después de un tiempo determinado.
- Limpiar imágenes antiguas generadas por la aplicación.
- Realizar operaciones en el log de transacciones de un archivo.

Para iniciar este servicio hay que incluir las siguientes líneas en el archivo de propiedades principal:

```
ScheduleObject.n=
ScheduleIntervalMinutes.n=
ScheduleApplication.n=
```

En *ScheduleObject* se indica el nombre de la clase del objeto Schedule, en *ScheduleIntervalMinutes* va el intervalo en minutos para ejecutar y en *ScheduleApplication* se indica el nombre de la aplicación donde se ejecutara el objeto. *n debe* ser remplazado por un número entre 1 y 20 que indica el número del *Schedule* a crear.

3.8.3.16. Jasper Reports

El framework tiene integración con *Jasper Report* para generación de reportes a través de una clase que hace uso de la información en los *DataStores*. Un reporte de este tipo puede ejecutarse en el cliente a través de un *Applet*, como una aplicación *Web Star*, o exportarse directamente a un archivo. Si se lo ejecuta como aplicación es parecido a lo siguiente:

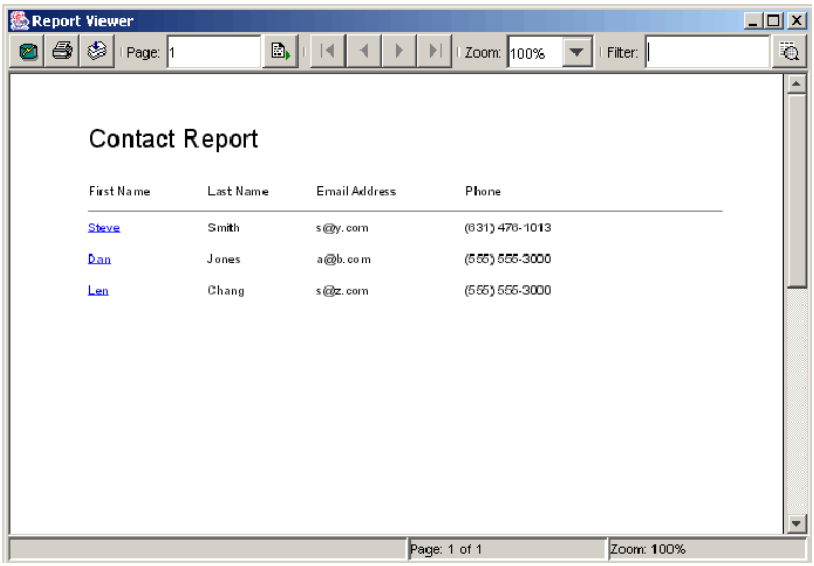


Figura 3.35: Reportes con Jasper
Fuente: Sofia User Guide

Para la generación de reportes se hace uso de paquete llamado *com.salmonllc.jasperReports*. Debido a que los reportes Jasper requieren acceso al sistema de archivos del cliente, los paquetes de datos pueden ser firmados digitalmente y esto se hace a través una utilidad llamada *jarsigner*.

La interface de un reporte mediante Jasper se la especifica mediante archivos XML los cuales deben ser escritos de acuerdo a un formato definido. Para evitarse escribir este código XML, estos archivos pueden generarse mediante una aplicación llamada *IReport.*

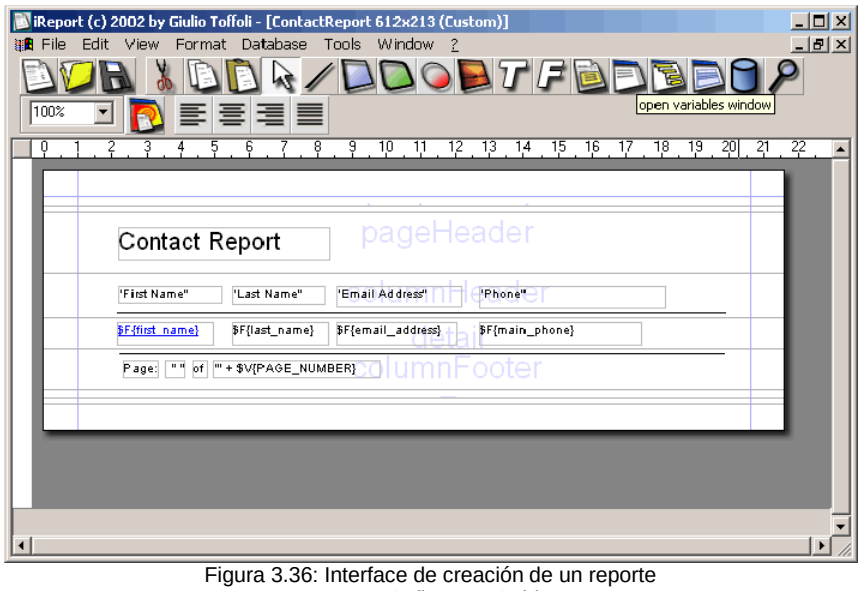


Figura 3.36: Interface de creación de un reporte
Fuente: Sofia User Guide

3.8.4. Campos de aplicación

SOFIA en su estado original es utilizado para el desarrollo de aplicaciones web, pero al ser código abierto permite integrarlo con otras herramientas para realizar aplicaciones más complejas en donde se puedan tener varios tipos de interfaces de usuario. Entre las empresas más conocidas que han trabajado directamente con SOFIA se encuentran[48]:

- Fujifilm: Se creó el sitio web totalmente en SOFIA para manejo de catálogos de productos y manejo de órdenes de ventas. Ha demostrado ser muy estable, ya que se ha registrado más de un millón de páginas vistas por día.
- American Finalcial Group: Se creó un sistema de inversiones que rastrea todas las transacciones hechas en la compañía (transacciones de bienes raíces o pagos de clientes).
- BNP Paribas: Sitio para manejo y control de deudas de clientes
- ASFA (ACS): Sistema de rastreo de prestamos a estudiantes.

[48] Salmon LLC. *Salmon LLC Clients*. Disponible en: <http://www.salmonllc.com/website/Jsp/vanity/Clients.jsp>

3.9. RESUMEN

SOFIA es una herramienta que intenta emular la funcionalidad y facilidad de desarrollo que ofrece .NET. Existen pocas buenas herramientas para crear aplicaciones Java para Web y SOFIA es una de estas según la comunidad de desarrollo. Una ventaja importante es su precio ya SOFIA no tiene costo y trabaja en conjunto con un grupo de herramientas de costo nulo, o muy bajo, en comparación con otras grandes herramientas de desarrollo.

Este framework implementa el patrón de diseño Model – View – Controller (Modelo, Vista, Controlador) para el desarrollo de aplicaciones. Trabaja en conjunto con Dreamweaver y Eclipse. Dreamweaver maneja la generación de las páginas JSP (la vista) y Eclipse permite la creación del controlador y el modelo.

La conexión a bases de datos a través de SOFIA es bastante sencilla, ya que solo se necesita el *JDBC* de la base y especificar los datos de acceso seguro (nombre de usuario, contraseña). Dentro de una aplicación se pueden tener varias conexiones a bases de datos a la vez; y se puede agregar y eliminar las conexiones en cualquier momento.

Cada aplicación trabaja con un archivo plano de propiedades que especifica la conexión a la base de datos, pero también propiedades de la interface gráfica como colores y fuentes, tal cual lo hace una hoja de estilos en cascada (CSS).

Los archivos y carpetas dentro de una aplicación pueden ser llamados mediante nombres lógicos en lugar de nombres reales, esto que permite modificar nombres de archivos si necesidad de cambiar las referencias a éstos dentro de la programación.

En Dreamweaver la creación de la vista es rápida, ya que para todo componente ofrece ventanas de ayuda donde se deben especificar sus características, esto permite que muchas veces no se necesite editar código cuando se crea una página JSP.

Cada componente de SOFIA se diferencia de un componente HTML normal en que pueden ser manipulados dentro del controlador de la página. Cada componente de SOFIA tiene el prefijo "*salmon*" para diferenciarlo de los normales. Se puede mezclar componentes SOFIA y HTML sin problema.

Eclipse permite generar el código base para la creación de los controladores y modelos, pero además ayuda en la escritura de nuevo código mediante ventanas de guía, funciones de auto-complemento de código, etc. Dentro de los controladores se maneja los eventos de la página (pulsación de botones o vínculos, cargado de páginas o cambios en datos de formularios).

El modelo permite la recolección de los datos desde la base de datos para que puedan ser manipulados por la vista o por el controlador. El modelo implementa funciones para edición, creación o borrado de datos; todo esto sin necesidad de escribir sentencias SQL, por lo que si se decide cambiar de base de datos en algún momento (por ejemplo de MySQL a Oracle) la aplicación no será afectada.

Las aplicaciones en SOFIA permiten implementar internacionalización. Esto permite que se muestren datos dependiendo el idioma de preferencia del usuario. Para esto se manejan archivos planos por cada uno de los idiomas.

Además se puede manejar *skins*. Esto que permite que una aplicación muestre colores y diseños de acuerdo a preferencias del usuario. Los *skins* son paquetes de configuraciones donde se especifican los colores y características de cada componente grafico del framework muy similar a lo que hace una hoja de estilo en cascada.

Existen componentes que tienen encapsulada toda la lógica por lo que el programador no se necesita preocupar por el funcionamiento, este es el caso de los formularios List, Detail y Search los cuales permiten el listado y edición tablas de datos sin necesidad de escribir líneas de código. Existen otros componentes avanzados como calendarios, barras de navegación o árboles de datos que son creados con tags dentro de la vista y permiten ser manipulados dentro de los controladores.

Capítulo 4

4. DESARROLLO DE APLICACIÓN PROTOTIPO

4.1. CASO DE ESTUDIO: ANÁLISIS DE PRECIOS UNITARIOS

4.1.1. Introducción

Para demostrar el proceso de desarrollo de aplicaciones en SOFIA, este documento especifica la creación de una aplicación prototipo semifuncional que simulará un *Sistema de Análisis de Precios Unitarios para Empresas Constructoras*. Mediante esta aplicación se demostrará las funcionalidades del framework para desarrollo de aplicaciones en web, tratando de utilizar la mayor parte de los componentes que ofrece.

Un sistema de análisis de precios puede ser usado para varios fines dentro del mundo de los negocios y las empresas. Toda empresa en algún momento necesita hacer procesos de análisis económicos para determinar la viabilidad de un negocio o para asignar el precio adecuado a un producto. El avance tecnológico provoca una constante evolución del mercado de los negocios haciendo que los tiempos de respuesta se reduzcan, y obligando a mejorar la productividad, por lo cual es importante automatizar la mayor cantidad posible de procesos dentro de las empresas[49].

[49] López Aguilar, Juan José. *Análisis de Precios Unitarios*. Disponible en: <http://www.monografias.com/trabajos6/anpre/anpre.shtml>

La aplicación de estudio se concentrará en un sistema para uso de *empresas constructoras* que sea utilizado en web y que permita el análisis de precios unitarios y la creación de presupuestos para obras de construcción. Toda empresa de construcción realiza estos análisis antes de intervenir en un proyecto, esto permite a estas empresas participar en licitaciones y concursos públicos, un proceso muy común en el mundo de las construcciones.

Se entiende por presupuesto de una obra o proyecto la determinación previa de la cantidad en dinero necesaria para realizarla. Para crear estos presupuestos se toma en cuenta la experiencia adquirida en otras construcciones de índole semejante. Las formas o métodos para realizar esa determinación son diferentes según sea el objeto que se persiga.

Estos análisis permiten determinar si el costo de una obra guarda la debida relación con los beneficios que de ella se espera obtener. Además se hacen estudios como base para financiar la obra, detallando información básica como unidades de medida y precios unitarios, y tomando en cuenta no sólo el precio de los materiales y mano de obra, sino también las circunstancias especiales que pueden aparecer en el proyecto. Esto obliga a penetrar en todos los detalles y a formar precios unitarios partiendo de sus componentes más básicos.

Dentro del análisis de precios intervienen varios factores como: a) costos indirectos tanto de operación como de obra, los cuales son costos que se consideran debido a imprevistos; b) factores de costo del mercado; c) costos directos, donde se manejan

los costos de materiales y equipos que intervienen en la construcción de una obra pero también se toman en cuenta factores como la depreciación de equipos; d) costos de mano de obra mediante análisis de los salarios; e) la utilidad que pretende obtener la empresa; f) y finalmente el rendimiento e integración de materiales y equipos. Todo esto con el fin de sacar con la mayor exactitud posible el valor total de un presupuesto[50].

Todos los factores enumerados anteriormente son evaluados dentro de los llamados *rubros* que no son más que pequeñas tareas dentro de una obra. Estos rubros son evaluados uno a uno y son dependientes directos de la mano de obra, materiales, transporte y equipos. De cada rubro se obtiene su costo total para posteriormente incluirlos en una *tabla de cantidades y precios* la cual calcula el precio total de la obra.

Se necesita un estudio muy extenso para entender todos los conceptos que se manejan en la creación de análisis de precios unitarios, pero, ya que este capítulo solo pretende demostrar el funcionamiento de SOFIA, solo se manejarán los conceptos generales y de mayor difusión.

4.1.2. Objetivos

El objetivo principal de un sistema de análisis de precios es la obtención de presupuestos para obras o proyectos. Se necesita conocer el proceso de creación de

[50] García, Carlos. *Ajuste y reconsideración de precios en la construcción*. Disponible en: <http://www.monografias.com/trabajos11/ajus/ajus.shtml>

los presupuestos dentro de las empresas constructores, y conocer cuales son las necesidades y metas dentro de la empresa.

Se deben tomar en cuenta varios conceptos utilizados en la construcción como *tareas* o *rubros*. Igualmente se manejan variables como *materiales*, *mano de obra*, *equipos* y *transporte*. Para ello se necesitan conocer los fundamentos de costos desde un punto de vista económico.

El sistema debe manejar formularios para ingreso y manipulación de información, además generación de reportes para presentación de datos finales, todo esto dentro de de un ambiente web, para ello el uso de una base de datos es imprescindible por lo que se necesita la creación de un modelo conceptual de datos.

4.1.3. Conceptos básicos en el Análisis de Precios

El análisis de precios unitarios no puede estar desligado de la contabilidad. En la actualidad la contabilidad no está comprendida como un conjunto de hechos referidos al pasado, sino que en muchos casos prevé situaciones, siendo su información congruente, por lo que resulta ser una eficaz ayuda a la administración. De ahí que resulte necesario conocer las definiciones básicas que se maneja.

4.1.3.1. Empresa y Cliente

Una empresa es la institución que ofrece productos o servicios de alguna clase, y por tanto recibe remuneraciones. Una empresa constructora es la que oferta a un cliente servicios profesionales o provee de recursos humanos, físicos y económicos para la

construcción de acuerdo a tiempos y requerimientos del contratante. Es llamado también contratista.

Un cliente es la persona natural o jurídica que requiere de servicios profesionales para la construcción de una obra en particular. Normalmente una institución hace llamado a una licitación para que varias empresas participen de acuerdo a normas, requisitos y formatos. Las ofertas de cada empresa son evaluadas y gana la mejor. La institución que hizo el llamado a la licitación se convierte en el cliente. Un cliente normalmente es de 2 tipos, los que pertenecen al sector privado (personas o instituciones) y los que pertenecen al sector público (ministerios o alcaldías). Es llamado también contratante.

Una vez que el contratante llama a licitación, la empresa contratista presenta su oferta, posteriormente el contratante analiza estas ofertas y contrata a la empresa cuya oferta sea la más adecuada.

4.1.3.2. Proyectos y obras

Los proyectos son actividades en estudio o planeamiento de las cuales una empresa desea participar.

Cuando una empresa ha sido contratada o ganó una licitación para realizar un proyecto determinado, entonces este proyecto se convierte en la obra. Por tanto la obra es una actividad en ejecución.

4.1.3.3. Costo directo e indirecto

La palabra costo tiene varios significados en función de muchas circunstancias, pero por ejemplo si se considera una fábrica que adquiere materia prima para incorporarla al producto final entonces el costo de estos productos es simplemente el precio que se ha pagado por ellos en el mercado. En conclusión, el costo es el valor que representa el monto total de lo invertido (tiempo, dinero y esfuerzo) para comprar o producir un bien o un servicio[51].

4.1.3.3.1. Costo directo

Costo directo es la suma de los costos unitarios de los renglones de materiales, equipos, transporte y mano de obra en el análisis de precios. Son aquellos gastos que tienen aplicación a un producto determinado.

Materiales	Son aquellos insumos consumibles ó instalables que quedan incorporados a la obra. Este renglón no es afectado por el rendimiento.
Equipos	Son aquellos insumos (maquinarias, equipos, herramientas, implementos) requeridos para la construcción y la terminación de la obra, etc. Estos pueden ser propios ó alquilados. Este renglón es afectado por el rendimiento. Los equipos se dividen en 2 tipos: pesado (maquinaria de excavación) y liviano (palas, martillos carretillas).
Mano de obra	Es la cantidad de personas especializadas, necesarias para realizar una labor determinada en un tiempo específico. La unidad de medida puede ser de dos tipos: la cantidad del tiempo en días o el número de horas hombres a emplearse. Se debe reflejar el porcentaje de prestaciones sociales, bonos y cualquier otro valor que se le pague al personal.
Transporte	Recursos utilizados para el transporte de materiales y equipos dentro de una obra. Para este costo se considera la distancia recorrida aplicando una tarifa por unidad.

Tabla 4.1: Elementos que forman los costos directos dentro del análisis de precios
Fuente: Monografía de Ajustes y reconsideraciones de precios para la construcción
Autor: Ing. Carlos García Carrasquero

Cada uno de estos ítems es evaluado dentro de un rubro para obtener los costos directos de este. Cada ítem se necesita evaluar en función de su cantidad y precio dentro de cada tarea.

[51] García, Carlos. *Ajuste y reconsideración de precios en la construcción*. Disponible en: <http://www.monografias.com/trabajos11/ajus/ajus.shtml>

4.1.3.3.2. Costo indirecto

Es un gasto que no puede tener aplicación a un producto determinado. El costo indirecto es el costo adicional al costo directo. Es la suma total de los gastos y beneficios que se agregan al costo directo, no contenido en éste, hasta integrar el precio total de venta. A estos costos normalmente se le suma de gastos técnicos (administrativos) necesarios para la correcta realización de la obra[52].

Un costo indirecto puede estar compuesto de los siguientes factores:

Rendimiento	Es la cantidad de trabajo realizado entre el tiempo empleado, cada organismo, empresa e institución tienen calculados los rendimientos de acuerdo a sus propias experiencias. Los mismos son en base a la cantidad de la unidad de medida de la partida entre el tiempo en días, por ejemplos: (M3/día), (M2/día), (KG/día), etc.
Utilidad	Representa el beneficio que debe obtener una empresa por haber invertido tiempo, dinero y trabajo. El mismo se expresa como un porcentaje de la inversión realizada (costo directo + gastos de administración). Este porcentaje puede variar entre empresas dependiendo del beneficio que quiera obtener cada una, así como también de la complejidad ó sencillez de la obra.
Imprevistos	Costo adicional que se agrega a razón de imprevistos que se pueden presentar dentro de la ejecución de la obra.
Depreciación	Distribución del costo de un activo a los largo de su vida útil.

Tabla 4.2: Elementos que forman los costos indirectos dentro del análisis de precios
Fuente: Monografía de Ajustes y reconsideraciones de precios para la construcción
Autor: Ing. Carlos García Carrasquero

4.1.3.4. Rubros y tablas de precios

Un rubro es un trabajo específico dentro de la ejecución de la obra que puede medirse, y para lo cual se estipula un precio. Dentro del proyecto existen rubros cuya incidencia es importante en la obra global ya sea por su costo, por su dificultad técnica o por el tiempo que representa su ejecución. Cada tipo de construcción tiene un conjunto básico de rubros[53].

[52] García, Carlos. *Ajuste y reconsideración de precios en la construcción*. Disponible en: <http://www.monografias.com/trabajos11/ajus/ajus.shtml>
[53] García, Carlos. *Ajuste y reconsideración de precios en la construcción*. Disponible en: <http://www.monografias.com/trabajos11/ajus/ajus.shtml>

Cada rubro tiene una descripción específica y especificaciones técnicas que lo identifica de los demás, además una unidad de medida para ser cuantificado y es común que en determinadas instituciones se les asigne un código para facilitar su manejo. Los rubros son los elementos constitutivos de una obra y son la base de su estudio económico. Además el contratante los utiliza como mecanismo de comparación entre las diferentes presupuesto.

El costo de construcción de cada unidad de rubro es función del costo de equipo, mano de obra, materiales y transporte; sumándole a esto los gastos efectuados para su ejecución y demás costos indirectos.

La tabla de cantidades y precios es donde se especifican los diferentes rubros que se utilizarán en la ejecución y terminación de la obra. La obra en su totalidad se la divide en trabajos, y dentro de cada trabajo se especifican un conjunto de rubros.

Dentro de esta tabla se especifican los rubros con cantidades, unidades de medida y costos, para posteriormente calcular el costo total de la obra[54].

4.1.3.5. Propuesta

La propuesta de una empresa contratista consta de la elaboración del presupuesto análisis de los precios unitarios para todas las actividades que se realicen dentro de la obra.

[54] López Aguilar, Juan José. *Análisis de Precios Unitarios*. Disponible en: <http://www.monografias.com/trabajos6/anpre/anpre.shtml>

Dentro de una empresa de construcción se manejan los *presupuestos* de proyectos que son el cálculo anticipado, en una fecha dada, del costo de una obra o de parte de ella a partir de especificaciones de construcción dadas. Este presupuesto es elaborado por el constructor, haciendo un seguimiento de cada una de las etapas de la obra. El presupuesto consta de varias partes como por ejemplo el nombre, unidades de medida, cantidades de obra a realizar, valor unitario por obra y el análisis de precios unitarios.

4.1.4. Necesidades y requerimientos

Para participar en una licitación de una obra normalmente se requiere que cada empresa participante presente un análisis completo de precios junto con el presupuesto de la obra.

El análisis de precios es muy extenso, ya que se requiere tener un registro de todos los ítems que conforman los costos directos con sus respectivos costos del mercado y unidades de medida. Después se necesitan construir los rubros que son conformados por uno o varios ítems (transporte, mano de obra, equipos y materiales) y cuya creación puede ser una tarea larga y tediosa. Además el número de rubros requeridos en una obra puede ser bastante extenso, y normalmente se los arma de forma manual. Normalmente todo esto una empresa lo hace mediante hojas electrónicas, por lo que toma gran cantidad de tiempo realizarlo.

Una empresa requiere tener un sistema que maneje bases de datos de todos los ítems utilizados en el rubro, además es necesario tener una base de rubros para no tener que

construirlos por cada nuevo análisis, sino que simplemente se debería utilizar los creados previamente y armarlos dentro de la tabla de cantidades y precios.

Los cálculos deberían ser automáticos al igual que la presentación de reportes. Se deben también poder revisar todos los recursos creados previamente para lo que se necesita tener un historial de proyectos.

Además es necesario que se autentifiquen usuarios para que solo usuarios registrados tengan acceso a la información.

4.2. METODOLOGÍA DE DESARROLLO PARA CASO DE ESTUDIO

Desarrollar sistemas informáticos no es fácil, requiere de varias etapas de estudio antes implementar un sistema. Siempre se deben estudiar las necesidades y metas del cliente. Debido a que la intención principal de la tesis no es el estudio de las diferentes etapas en el desarrollo de aplicaciones, sino el análisis de herramientas de desarrollo, se hará simplemente un breve análisis de las etapas de desarrollo. Por esta causa, el análisis referente a aspectos técnicos del desarrollo de aplicaciones será tomado de manera superficial, aunque por su naturaleza requieran de un análisis más específico propio de la Ingeniería del Software.

Se requiere entonces definir las etapas para este caso de estudio, así como las plataformas de desarrollo y metodología básica a seguir. La plataforma de desarrollo debe obedecer a las necesidades de la empresa y a la proyección que se tenga en el uso de nuevas tecnologías. Con motivo de demostrar la funcionalidad de SOFIA, la aplicación será realizada para ambientes web con acceso a una base de datos relacional.

Todas las etapas de desarrollo que involucran un sistema (la estrategia, el análisis, el diseño, la construcción, la documentación y la implementación entre otras) serán resumidas de manera práctica en tres etapas fundamentales: el análisis, el diseño y la implementación; las cuales son basadas en el ciclo de vida de los sistemas. No se adentrará en características más específicas del desarrollo como puede ser manejo de versiones, control de cambios o pruebas del sistema.

4.2.1. Análisis

Dentro de esta sección se especificará el ciclo de vida del sistema, cuales son sus requerimientos funcionales y las necesidades que tiene cliente en la actualidad.

4.2.1.1. Investigación preliminar

Un sistema de análisis de precios es de gran importancia dentro de la propuesta que hace una empresa para la ejecución de una obra de construcción. Por esto se debe conocer cuales son los procesos que se sigue para la creación estos análisis.

Para la creación del análisis de precios unitario la base fundamental son los recursos. Estos recursos son utilizados una y otra vez dentro del sistema. Existen 4 tipos: *equipo*, *materiales*, *transporte* y *manos de obra*.

Cuando se va a crear una nueva propuesta, los datos (en especial el precio) de cada uno de estos recursos son actualizados a la fecha. Además a algunos de estos recursos se les aplica unidades de medida y a otros un rendimiento.

Para los equipos y herramientas normalmente se necesita una *tarifa/hora* y rendimiento. Los materiales tienen asignado una *unidad de media* y un *precio por unidad*. El transporte también tiene asignado una *unidad de medida* y una *tarifa por unidad de medida*. Por último la mano de obra tiene *salarios* asignados y se aplica un *rendimiento*.

Cuando cada uno de estos datos se actualiza, entonces se crean los *rubros* que son utilizados para la ejecución de la obra. Cada rubro debe ser vinculado a la empresa contratista y al proyecto. Normalmente el rubro se identifica con un nombre único pero también se le puede asignar un código. Otros datos importantes son la unidad de medida y la descripción. Todos estos valores normalmente van en la cabecera del rubro.

Posteriormente se agrega al rubro cada uno de los recursos necesarios, separados por tipos:

- El equipo toma los valores de tarifa pero además se le asigna cantidad y rendimiento para calcular el *costo total de equipo*.
- Para los materiales el precio y la cantidad son también necesarios, aunque también se especifica la unidad de medida, y se saca el *costo total de materiales*.
- Para el transporte se necesita de la unidad, la cantidad, la tarifa por unidad, la distancia recorrida, y con todo esto se calcula el *costo total de transporte*.
- Para calcular el *costo total de mano de obra* son necesarios la cantidad, el salario y el rendimiento.

La suma de todos estos costos forma el *total de costos directos*.

A cada rubro se le asigna un porcentaje que corresponde a costos *indirectos y utilidad*. Este porcentaje puede ser diferente por cada rubro, pero en la mayoría de los casos el valor es asignado a todos los rubros del proyecto por igual. Finalmente se

calcula el *costo total del rubro* sumando el total de costos directos más los costos indirectos y la utilidad.

Posteriormente se crea la *tabla de cantidades y precios* donde se indican los diferentes rubros que son parte de la obra final. Cada rubro dentro de la tabla de análisis de precios debe poseer su número, su código, su nombre o descripción, su unidad de medida, una cantidad asignada por cada trabajo y el precio final del rubro. De la suma de estos valores se obtiene el *precio total del proyecto*.

4.2.1.2. Requerimientos del sistema

El sistema debe automatizar cada uno los procesos descritos en la sección anterior.

Primero el sistema debe constar de una base de datos que almacene cada uno de los tipos de recursos y, mediante interfaces amigables al usuario, se debe poder editar sus valores (precio, unidad de medida). También debe permitir crear nuevos recursos en cualquier momento.

Cuando se va a crear un nuevo análisis de precios primero se debe ingresar los datos del cliente para vincularlos a la tabla de cantidades y precios.

También debe tener una base de datos de rubros para no tener que crearlos una y otra vez con cada nuevo proyecto. Pero igualmente debe permitir crear nuevos rubros si no existiera uno requerido. Para la creación de rubros, los datos de los recursos deben se recuperados de la base de datos. Además el sistema debe hacer los cálculos

correspondientes para la obtención de los costos totales sean estos directos o indirectos, y finalmente calcular el costo total del rubro.

Si los rubros ya están creados, dentro de la tabla de cantidades y precios se los debería cargar desde la base de datos junto con los datos relevantes. Solo se necesitaría asignar cantidades a cada rubro para que el sistema calcule el costo total de la obra.

Finalmente el sistema debería estar en la capacidad de generar reportes de los rubros y de las tablas de cantidades y precios dentro de un proyecto.

4.2.2. Diseño

Dentro del diseño se indicarán los modelos de base de datos para el sistema, el formato de la interface gráfica, y los diferentes procesos que se manejan dentro de la creación del análisis de precios.

4.2.2.1. Base de Datos

El modelo E/R presentará el diagrama lógico y físico la base de datos del sistema. Aquí se especifica cada una de las tablas que servirán de medios de almacenamiento de los datos.

El diseño de la base de datos se lo tratará de simplificar lo más posible para que cumpla los requerimientos mínimos de funcionalidad de un análisis de precios:

A continuación se detallan las tablas más importantes existentes en la base de datos:

Tabla	Función
Usuario	La tabla de usuarios contiene información personal de usuarios del sistema. Esta tabla sirve para validar el ingreso de usuarios. Deber almacenar un seudónimo y una contraseña. No se manejan roles dentro de la aplicación. Un usuario tiene acceso a cualquier parte del sistema sin excepción.
Proyecto	Es una tabla que contiene información básica del proyecto específico sea esto nombre y fecha de entrega. Además a un proyecto se le debe asignar una empresa (contratista) y un cliente (contratante). Para esto existen 2 tablas que almacenan la información básica de clientes y empresas (nombre y RUC)
Presupuesto	Esta tabla almacena los presupuesto creados por cualquier usuario. Un presupuesto está en directa relación con un proyecto, por lo que si no se crea un proyecto, no se podrá crear un presupuesto
Unidad de Medida	La unidad de medida es simplemente una tabla que amacena el nombre y el símbolo de un tipo de medida dado
Recursos	En realidad existen cuatro tablas de recursos: Mano de obra, Equipos, Materiales, Transporte. Cada tipo de recurso se lo almacena en tablas separadas, esto se hace porque algunos recursos necesitan unidad de medida y otros no. Todos estos recursos necesitan almacenar el nombre, una descripción y un precio
Rubro	Esta tabla almacena información de cabecera de un rubro y mediante tablas intermedias se vincula el rubro a uno o varios recursos previamente creados.

Tabla 4.3: Tablas de la Base de Datos
Autor: Edison Hernández

Revisar ANEXO 1

4.2.2.2. Procesos del sistema

Se indican los datos de entrada, los que serán calculados y los que deben ser almacenados. Además se especifica la estructura de archivo y los dispositivos de

almacenamiento. Los procedimientos que se escriben indican cómo procesar los datos y producir salidas.

Mediante el uso de diagrama de flujos de datos se estudia los procesos que intervienen para llevar a cabo una tarea específica dentro del sistema.

Este diagrama da un panorama que muestra las tareas realizadas y como son hechas por el sistema. Incluyen nombres de personas o roles, nombres de departamentos, archivos, equipo, dispositivos utilizados, ubicaciones, nombres de procedimientos y medios de almacenamiento.

En vista de que este sistema solo maneja un tipo de rol (Usuario Administrador), los diferentes roles detallados en los diagramas de flujos pueden ser manejados por una sola persona.

Revisar ANEXO 2

4.2.2.3. Diseño de pantallas

Se indica un esquema pantalla que sea similar al sistema terminado, aquí se especifica cada una de las pantallas que maneja la aplicación.

Revisar ANEXO 3

4.2.3. Construcción

En esta sección se indica el ciclo de vida que convierte los modelos y diseños en el software final. Esto implica la codificación de las unidades del programa, la generación automatizada del código, y el montaje de los componentes reutilizables ya construidos y probados.

4.2.3.1. Estándares de desarrollo

Se especifican estándares tanto para nombres de archivos como para la programación.

Interfaces	**Estándar**	<TIPO><NOMBRE>
	Tipo	btn Button chk Check Box frm Form hdn Hidden Field lbl Label lnk Link lst List img Image mnu Menu pwd Password Field rdb Radio Button rdg Radio Group tbl Table txt Text ted Text Edit tea Text Edit Area
	Ejemplo	btnPersonGrabar txtEstandar
Archivos	**Estándar**	<NAME>.<FILETYPE>
	Ejemplo	EntidadView.html EntidadListView.jsp
Variables	**Estándar**	<SCOPE><TYPE><NAME> <SCOPE><SETTYPE><TYPE><NAME>
	Alcance	a... Application s... Session l... Local p... Parameter
	Tipo de dato	...x<NAME> Blob ...b<NAME> Boolean ...d<NAME> Date ...c<NAME> Decimal ...u<NAME> Double ...i<NAME> Integer ...l<NAME> Long ...s<NAME> String

		...t<NAME> Time ...r<NAME> Real
	Tipo de objeto	...a<TYPE><NAME> Arreglo ...l<TYPE><NAME> Lista ...o<TYPE><NAME> Objecto
	Nombre	...<Singular Object><Singular Atributte> ...<Singular Object><Singular Method>
	Ejemplo	aiEntidadContador aaiEntidadId

Tabla 4.4: Estándares de programación
Autor: Edison Hernández

4.2.3.2. Implementación

La implementación de la aplicación se la hace a través de SOFIA, en conjunto con las herramientas integradas (Eclipse y Dreamweaver). Esta aplicación trata de regirse a con todos los estándares indicados en las secciones anterior.

La manipulación de datos se la hace a través de los componentes de SOFIA, tratando de utilizar su funcionalidad al máximo para demostrar todas las prestaciones, pero también las deficiencias que poseen.

Se hace uso de formularios de búsqueda, detalle y listado para la manipulación de los datos, también se hace uso de barras de navegación y árboles para el manejo de las opciones y menús.

Otro aspecto importante utilizado en la aplicación es el uso de los *skins* para personalizar la interface de usuario. A través de esto, junto con archivos de propiedades y de mapas del sitio se especifica el formato en el que se mostrará la aplicación. Se trata de dar el mayor uso posible a cada uno de las ventajas que ofrece el framework.

El manejo de mensajes informativos y errores se maneja a través de clases. Las validaciones de datos y detección de errores se las hace a través de los modelos de datos. Además se maneja los archivos de *logs* para el sistema, estos archivos se manejan con los formatos definidos el framework.

El código fuente de la aplicación será entregado junto con este documento. Para mayor información referirse a estos.

Capítulo 5

5. CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

Se realizó un estudio de las diferentes propuestas del mercado para el desarrollo de aplicaciones en internet haciendo énfasis en el framework SOFIA.

Se hizo un estudio comparativo de las principales herramientas IDE (Integrated Development Tool) para Java que existen en el mercado. Se destacó Eclipse como la herramienta más cómoda y versátil del mercado. Cómoda, ya que es una herramienta "Open Source" gratuita; y versátil porque permite la inclusión de extensiones adicionales. Además posee gran soporte y fiabilidad, ya que detrás está la reconocida empresa IBM que, de alguna manera, es la creadora y promotora de Eclipse.

Se conoció los fundamentos, características y usos de SOFIA para el desarrollo de aplicaciones. SOFIA demostró ser una herramienta muy útil y fácil para trabajar con Java y JSPs. Su instalación es muy simple al igual que su uso. Viene con un ambiente para la creación de la capa de presentación y otro para la capa de negocio. Esto permite separar totalmente estas dos etapas del desarrollo y permite asignar diferentes equipos de trabajo.

La separación de las diferentes capas de una aplicación tiene grandes ventajas. No se mezcla la lógica del negocio con la interfaz de usuario (la programación de la página y su funcionalidad no debería estar incluida junto con la vista) lo que es de gran

ventaja a la hora del mantenimiento y las hace totalmente independientes. El modelo (Model) y el controlador (Controller) corresponden a la etapa de programación de una aplicación. Mediante SOFIA y Eclipse se agilita este proceso, ya que genera el código necesario para la creación de cada uno de estos componentes. La creación de la vista (View) es muy sencilla utilizando una de las principales herramienta de desarrollo web (Dreamweaver) que, mediante la generación de código, acelera todo el proceso de desarrollo de una aplicación.

SOFIA presenta varios componentes que en realidad ayudan al programador a enfocarse en la lógica del negocio en lugar de detalles de funcionamiento que nada tiene que ver con la aplicación. Estos componentes aceleran el desarrollo pero tienen sus desventajas, ya que la mayoría de ellos no pueden ser modificados en su funcionamiento y estructura. Muchos de estos componentes se crean mediante ventanas de ayuda que permiten editar sus atributos básicos, pero existen otros atributos que no pueden ser alterados, haciéndolos poco flexibles. La estructura visual es otro de los problemas que presentan estos componentes, ya que tienen una forma de mostrarse que normalmente no puede ser modificada (por ejemplo como muestran datos, botones, colores, bordes, etc.) lo que es un obstáculo para la creación de la interfaz de usuario. A pesar de todo, la funcionalidad que ofrecen cada uno de estos componentes normalmente es suficiente para manejar tareas importantes dentro de una página web.

En mi experiencia personal no he visto una herramienta como esta para Java. Inclusive en otros lenguajes de programación como PHP no existen herramientas que

permitan agilitar el desarrollo como lo hace SOFIA. En PHP por ejemplo, el desarrollador debe ocuparse tanto de la vista como de la lógica del negocio, ya que estas se encuentran unidas en un solo documento, además obliga al desarrollador a ocuparse de detalles y funcionalidades que no tienen nada que ver con la aplicación en sí. SOFIA cubre estos aspectos.

En general SOFIA se puede considerar como una herramienta para un rápido desarrollo de aplicaciones web, no muy flexible pero bastante funcional. Es posible crear una simple página con integración a base de datos sin ni siquiera escribir líneas de código, lo que la hace muy intuitiva a la hora de utilizarla.

Con respecto a la base de datos, se ha comprobado que MySQL es una base de datos muy confiable y rápida, además de ser muy sencilla de utilizar. La conexión de MySQL con SOFIA es muy simple.

Para la creación la aplicación semifuncional de análisis de precios unitarios se trató de aplicar todas las características que ofrece SOFIA en las diferentes etapas del desarrollo. Además se hizo un análisis bastante general de lo que implica la creación de una aplicación de este tipo.

El análisis de precios unitarios ha demostrado ser unos de los procesos fundamentales de los que hace uso una empresa de construcciones. Se trató de crear un sistema para web que permita la administración y control de las diferentes etapas que componen el análisis de precios a pesar que se manejan varios conceptos de

construcción y modelos matemáticos, que pueden llegar a ser complicados de entender.

De acuerdo a criterios de algunos expertos en la materia, aplicaciones de este tipo no ameritan estar disponibles en internet, ya que normalmente el análisis de precios se lo hace en la oficina de la empresa, en grupos de trabajo y con información de varias fuentes. Por esto una aplicación de tipo Cliente/Servidor de 2 capas podría ser lo más adecuado, pero una aplicación en web no, a menos que se la implemente en una Intranet de una oficina.

El desarrollo de esta tesis me sirvió para entender mejor en que consiste la plataforma J2EE, varios conceptos del desarrollo web, pero sobre todo me sirvió para entender el manejo SOFIA. Además fue útil para comprender la utilidad del análisis de precios unitarios, ya que es fundamental en una empresa de construcción.

5.2. RECOMENDACIONES

A pesar de tener ventajas, las aplicaciones web suelen ser menos amigables para el usuario que las cliente/servidor porque las vista son más difíciles de crear y tienen menos funciones. Hay que investigar a fondo las necesidades del cliente a la hora de escoger la herramienta de desarrollo adecuada, y si no es fundamental que la aplicación esté en la web; cliente/servidor es una opción válida hoy en día.

Tomar en cuenta que el desarrollo de aplicaciones de tres o más capas simplifica el mantenimiento de una aplicación, por tanto siempre que sea posible es recomendable utilizar este tipo de arquitectura.

Se recomienda hacer uso de la última versión disponible de SOFIA, ya que constantemente se liberan nuevas versiones de esta herramienta, en las que se pueden encontrar importantes mejoras con respecto a las versiones anteriores.

Las características técnicas de las computadoras para desarrollo deben ser elevadas. Por experiencia personal se recomienda una computadora Pentium IV con no menos de 512 MB de memoria, para poder levantar el servidor Tomcat, que puede tardar más de un minuto, y junto con las otras herramientas que deben estar abiertas para que funcione el framework, pueden ocupar gran parte de los recursos de la máquina.

El uso de MySQL como base de datos es recomendable cuando se tienen poca cantidad de registros, ya que sacrifica seguridad por velocidad. Si se tuviera un gran

número de registros (100 mil o más), se ha comprobado que su desempeño disminuye notablemente, especialmente en búsquedas. Una alternativa a esta base de datos puede ser PostgreSQL o Firebird que también son motores libres e incorporan características que MySQL no implementa.

Las herramientas "Open Source" para desarrollo de aplicaciones están abriéndose espacio en el mercado con gran rapidez. El auge de herramientas gratuitas y de código abierto se ha vuelto cada vez mayor, y es una opción valida ante las herramientas comerciales que normalmente tienen costos elevados. Muchas de las herramientas libres ofrecen iguales o superiores prestaciones que las herramientas comerciales. Por esto, las herramientas libres merecen ser investigadas más a fondo, incluso en combinación con herramientas comerciales, como lo hace SOFIA.

Igualmente este tema se podría tratar como una nueva materia de la carrera de Sistemas. El uso de aplicaciones libres es hacia donde se está encaminando el mundo, actualmente para todo tipo de herramienta imaginable existen una opción "libre" que merece ser investigada.

Eclipse es una herramienta libre de gran utilidad. Se podría considerar una plataforma de trabajo, debido a la gran cantidad de opciones, configuraciones posibles y, sobre todo, extensiones adicionales que ofrece. Un estudio completo de la herramienta puede tomar algún tiempo por lo que podría considerarse como tema de disertación de grado para la carrera de sistemas.

SOFIA podría presenta mejoras necesarias para una mayor utilidad. Sería útil que este framework trabaje no solo en Windows sino también en otros importantes sistemas operativos como GNU Linux o Mac OSX. Otro aspecto importante es hacerla más flexible, ya que en especial los componentes gráficos no pueden ser cambiados en su estructura, lo que impide al diseñador grafico hacer su trabajo de mejor manera.

BIBLIOGRAFÍA

- **Introducción**

Lindholm, Tim; Yellin, Frank. 1999. *The JavaTM Virtual Machine Specification*. Disponible en: <http://java.sun.com/docs/books/vmspec/2nd-edition/html/Introduction.doc.html>. (Teoría sobre la Maquina Virtual de Java)

Salvatierra, Marlo. 1997. *Publicidad en Internet*. Disponible en: <http://www.monografias.com/trabajos7/puin/puin.shtml>

Venegas, J. 1999. *Herramientas de Creación Web*. Universidad de Valladolid. Disponible en: <http://www.infor.uva.es/~jvegas/cursos/buendia/pordocente/node10.html>

Venegas, J. 1999. *Introducción a las Aplicaciones Web*. Universidad de Valladolid. Disponible en: <http://www.infor.uva.es/~jvegas/cursos/buendia/pordocente/node11.html>. (Conceptos básicos de html)

Venegas, J. 1999. *Desarrollo de Aplicaciones Web*. Universidad de Valladolid. Disponible en: <http://www.infor.uva.es/~jvegas/cursos/buendia/pordocente/node17.html>. (Conceptos sobre Arquitectura y Servidores web.)

Mendez, Justo. 1997. *Lenguajes de Programación*. Disponible en: <http://www.monografias.com/trabajos/lengprog/lengprog.shtml>. (Conceptos básicos de lenguaje Java>

Nango Quintana, Carlos Ernesto. *Chat´s en el lenguaje Java*. Disponible en: <http://www.monografias.com/trabajos12/chtjava/chtjava.shtml>. (Conceptos Básicos De Java, Cliente/Servidor)

MySQL AB. 2004. *MySQL Licensing Policy*. Disponible en <http://www.mysql.com/company/legal/licensing/>. (Políticas de licenciamiento de MySQL)

Sun Microsystems. *Java Frequently Asked Questions*. Disponible en: <http://java.sun.com/j2se/faq.html#Licensing>. (Licenciamiento de JDK)

Galáz, Solange. *Ingeniería de software*. Disponible en: <http://www.monografias.com/trabajos5/inso/inso.shtml>. (Etapas de la ingeniera de software para desarrollo de proyectos)

Torres Lecuanda, Araceli. *Metodologías modernas de desarrollo de Sistemas de Información*. Disponible en: <http://www.monografias.com/trabajos12/docmento/docmento.shtml>

- **J2EE**

Armstrong, Eric; Ball, Jennifer; Bodoff, Stephanie; Bode Carson, Debbie; Evans, Ian; Green, Dale; Haase, Kim; Jendroc, Eric. 1996. *The J2EE 1.4 Tutorial*. Disponible en: <http://java.sun.com/j2ee/1.4/docs/tutorial/doc/>

Sun Microsystems; traducido por Juan Antonio Palos. *Introducción a J2EE*. Disponible en: <http://www.programacion.com/java/tutorial/j2ee/>

Sun Microsystems. 2003. *The J2EE Tutorial (A beginner's guide to developing enterprise applications)*. Disponible en: <http://java.sun.com/j2ee/tutorial/1_3-fcs/>

Sun Developer Network Community. *White Paper - JavaServer Pages Technology - White Paper*. Disponible en: <http://java.sun.com/products/jsp/whitepaper.html>. (Conceptos básicos JSP)

Sun Developer Network Community. *White Paper - The Java Servlet API White Paper*. Disponible en: <http://java.sun.com/products/servlet/whitepaper.html>. (Conceptos básicos Servlets)

Sun Developer Network Community. *White Paper - JavaServer Pages Technology - JavaServer Pages Comparing Methods for Server-Side Dynamic Content White Paper*. Disponible en: <http://java.sun.com/products/jsp/jspservlet.html>

Sun Developer Network Communit. *JDBC API Overview*. Disponible en: <http://java.sun.com/products/jdbc/overview.html>. (Conceptos JDBC)

Sun Developer Network Community. *White Paper - JavaServer Pages[tm] Technology - JavaServer Pages Servlet Developer White Paper*. Disponible en: <http://java.sun.com/products/jsp/jsp-developerwp.html>. (Introduction to JavaServer Pages)

Sun Developer Network Community. *White Paper - JavaServer Pages[tm] Technology - JavaServer Pages White Paper*. Disponible en: <http://java.sun.com/products/jsp/jspguide-wp.html>. (JSP vs. CGI)

Díaz, Moisés Daniel. 2003. *Diseño de aplicaciones Internet usando los Patrones de diseño J2EE*. Disponible en: <http://www.programacion.com/java/articulo/corej2ee_patterns/>

Molpeceres Touris, Alberto; Pérez Mariñán, Martín. Diciembre, 2002. Arquitectura empresarial y software libre, J2EE. Disponible en: <http://www.javahispano.org/articles.article.action?id=70>. (J2EE – Herramientas gratuitas)

Sheil, Humphrey; Monteiro, Michael. Junio, 2002. *Rumble in the jungle: J2EE versus .Net, Part 1*. Disponible en: <http://www.javaworld.com/javaworld/jw-06-2002/jw-0628-j2eevsnet.html>. (Conceptos J2EE).

Fung, Kam Hay. Febrero, 2003. *Standard for JSP pages*. Disponible en: <http://developer.java.sun.com/developer/technicalArticles/javaserverpages/code_convention/>. (Convenciones de Programación Java para Páginas JSP)

Alarcón Mery, Constantino Samuel. Agosto, 2004. *Aplicación de Patrones J2EE en un Caso de Estudio*. Disponible en: <http://www.programacion.com/java/articulo/inukisoft/>

Sun Microsystems. *Developing Enterprise Applications Using the J2EE Platform*. Disponible en: <http://java.sun.com/developer/onlineTraining/J2EE/Intro2/j2ee.html>

Sun Developer Network Communit. *JavaServer Faces Technology Overview*. Disponible en: <http://java.sun.com/j2ee/javaserverfaces/overview.html>

Sun Developer Network Communit. J2EE Connector Architecture. Disponible en: <http://java.sun.com/j2ee/white/connector.html>

Sheil, Humphrey. Diciembre, 2001. *To EJB, or not to EJB?*. Disponible en: <http://www.javaworld.com/javaworld/jw-12-2001/jw-1207-yesnoejb.html>

Sun Microsystems; traducido por Juan Antonio Palos Título. Diciembre, 2003. *Catálogo de Patrones de Diseño J2EE. Y II: Capas de Negocio y de Integración*. Disponible en: <http://www.programacion.net/java/tutorial/patrones2/.>

Chamorro Villar, Ricardo. Noviembre, 2002. *Análisis comparativo entre Microsoft® .NET y Sun® J2EE*. Disponible en: <http://www.ciberteca.net/articulos/programacion/net/>

Zhong Jian. Septiembre, 2001. *Step into the J2EE architecture and process*. Disponible en: <http://www.javaworld.com/javaworld/jw-09-2001/jw-0928-rup.html>.

Rodriguez Lasterra, Enrique. Julio, 2002. *Introducción a Enterprise Java Beans*. Disponible en: <http://www.javahispano.org/articles.article.action?id=63>. (Enterprise Java Beans)

Gutiérrez, Gustavo A. *J2EE. Una plataforma para el cómputo empresarial*. Disponible en: <http://www.enterate.unam.mx/Articulos/dos/junio/j2ee.htm>

Sun Microsystems; traducido por Juan Antonio Palos. Junio, 2001. *Beans (Básico)*. Disponible en: <http://www.programacion.com/java/tutorial/beans/>

Sun Microsytems; traducido por Juan Antonio Palos. Mayo, 2001. *JavaBeans Enterprise*. Disponible en: <http://www.programacion.com/java/tutorial/javabeans/>

Sun Microsytems; traducido por Juan Antonio Palos. Mayo 2001. *Servlets y JSP*. Disponible en: <http://www.programacion.com/java/tutorial/servlets_jsp/>

Sun Microsytems; traducido por Juan Antonio Palos. Enero, 2003. *Introducción a los Servicios Web en Java*. Disponible en: <http://www.programacion.com/java/tutorial/servic_web/>

Abián, Miguel Ángel. Mayo, 2002. *J2ee Y .Net: La Rivalidad Permanente*. Disponible en: <http://www.javahispano.org/articles.article.action?id=55>

- **IDE**

Sun Microsystems. Noviembre, 2003. *Sun Java System Studio Standard 5 update 1Data Sheet*. Disponible en: <http://wwws.sun.com/software/sundev/jde/datasheet.html>

Storkel, Scout; traducido por Juan Antonio Palos. Septiembre, 2004. *Eclipse -- I -- Historia y Toma de Contacto*. Disponible en: <http://www.programacion.net/java/articulo/jap_eclip_1/>

Abián, Miguel Ángel. Febrero, 2003. *EL ARCHIPIÉLAGO ECLIPSE*. Disponible en: <http://www.javahispano.org/articles.article.action?id=75>

PC Magazine. Mayo, 2003. *Sun ONE Studio*. Disponible en: <http://www.pcmag.com/article2/0,1759,1191823,00.asp>

Sun Microsystems. *Sun Java Studio Standard 5 update 1 Features & Benefits*. Disponible en: <http://wwws.sun.com/software/sundev/jde/features/index.html>

Stephens, Matt. Abril, 2004. *First Look: Sun Java Studio Creator (Early Access)*. Disponible en: <http://www.softwarereality.com/reviews/creator_ea.jsp>

- **Struts**

The Apache Software Foundation. *Documentación Struts*. Disponible en: <http://struts.apache.org/userGuide/index.html>

The Apache Software Foundation; traducido por Juan Antonio Palos. Marzo, 2002. *El API Struts*. Disponible en: <http://www.programacion.com/java/tutorial/struts/1/>. (Conceptos de Struts y MVC)

Antoniucci, Javier. Abril, 2002. *Manual Básico de Struts*. Disponible en: <http://www.programacion.com/java/tutorial/joa_struts>. (Introducción a Struts)

- **Salmon SOFIA**

Salmon LLC. *Salmon LLC – SOFIA (Salmon Open Framework for Internet Applications)*. Disponible en: <http://www.salmonllc.com/website/Jsp/vanity/Sofia.jsp>

Dubinsky Dan. *Project SOFIA Forum*. Disponible en: <http://sourceforge.net/projects/salmon>

Santiago, Carlos. Marzo, 2004. *How does your framework compare to SOFIA*. Disponible en: <http://mail-archive.objectweb.org/barracuda/2004-03/msg00021.html>. (Introducción a SOFIA)

Dueñas, J. C. *Modelo-Vista-Controlador*. Disponible en: <http://polaris.dit.upm.es/~jcduenas/patrones/Modelo.htm>. (Introducción y contexto de MVC)

Sun Microsystems. 2002. *Designing Enterprise Applications with the J2EETM Platform, Second Edition*. Disponible en: <http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/titlepage.html>. Capitulo 4.3, 4.4. (Web-Tier Application Structure, Web-Tier Application Framework Design)

Simal Gándara, Juan. *Desarrollo de Aplicaciones con tecnología Internet*. Disponible en: <http://www.soluziona.es/htdocs/areas/consultoria/interes/articulos/tecnologia_internet.shtml>. (Arquitectura de las aplicaciones en tres capas, MVC)

Manzanedo del Campo, Miguel Ángel; García Peñalvo, Francisco; Cruzado Nuño, José Ignacio. *Guía de Iniciación al Lenguaje JAVA*. Capitulo IV. Disponible en: <http://pisuerga.inf.ubu.es/lsi/Invest/Java/Tuto/IV_3.htm>. (Conceptos de SWING)

- **Análisis de Precios Unitarios**

López Aguilar, Juan José. Noviembre, 2000. *Análisis de Precios Unitarios*. Disponible en: <http://www.monografias.com/trabajos6/anpre/anpre.shtml>. (Conceptos dentro del análisis de precios)

Garcia Carrasquero, Carlos. Noviembre, 2002. *Ajuste y reconsideración de precios en la construcción*. Disponible en: <http://www.monografias.com/trabajos11/ajus/ajus.shtml>. (Conceptos básicos de análisis de precios)

Murillo Rountree, Gabriel. 1989. *Administración de Empresas Constructoras*. 1ra edición .Guayaquil, Ecuador.

Merino, Wilfredo. 1979. *Manual de composición de costos unitarios para la construcción de carreteras*. Editorial Politécnica.

ANEXOS

ANEXO 1: MODELO ENTIDAD – RELACIÓN

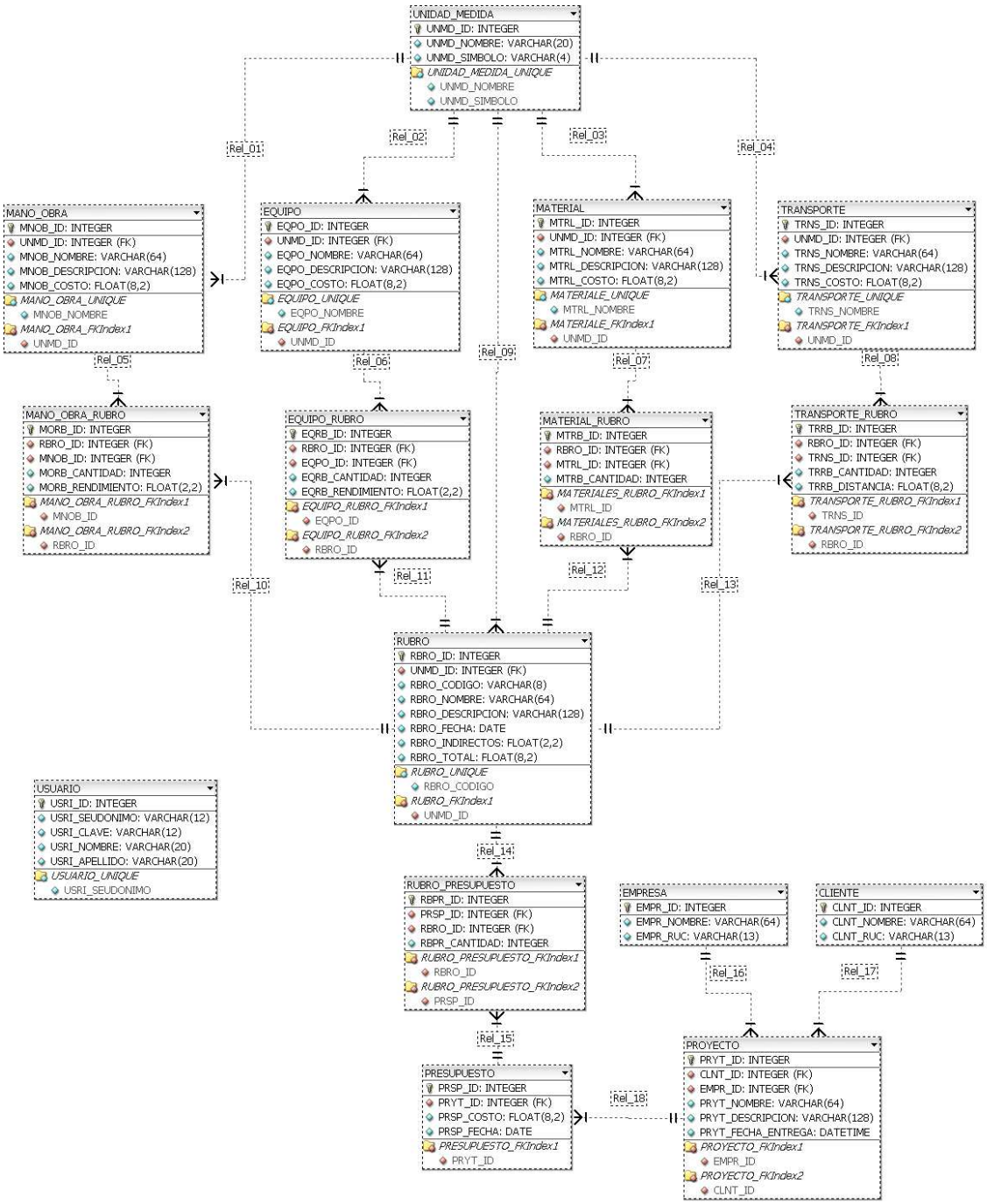


Figura A1.1: Modelo conceptual de base de datos
Autor: Edison Hernández

ANEXO 2: FLUJO DE PROCESOS

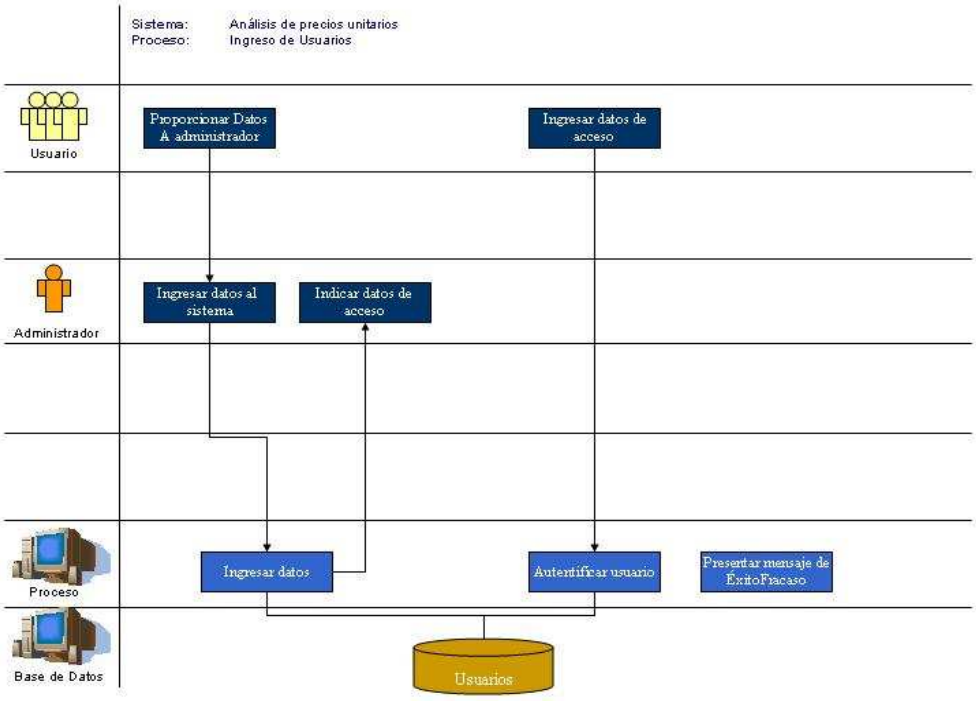


Figura A2.1: Flujo de procesos – Autentificación de usuarios
Autor: Edison Hernández

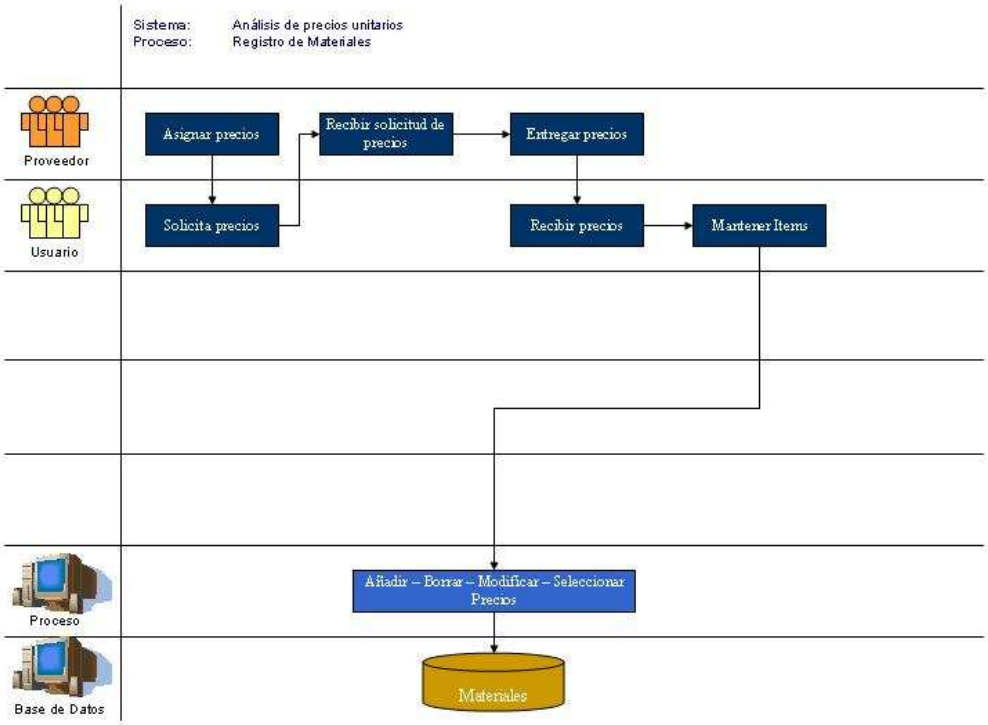


Figura A2.2: Flujo de procesos – Registro de materiales
Autor: Edison Hernández

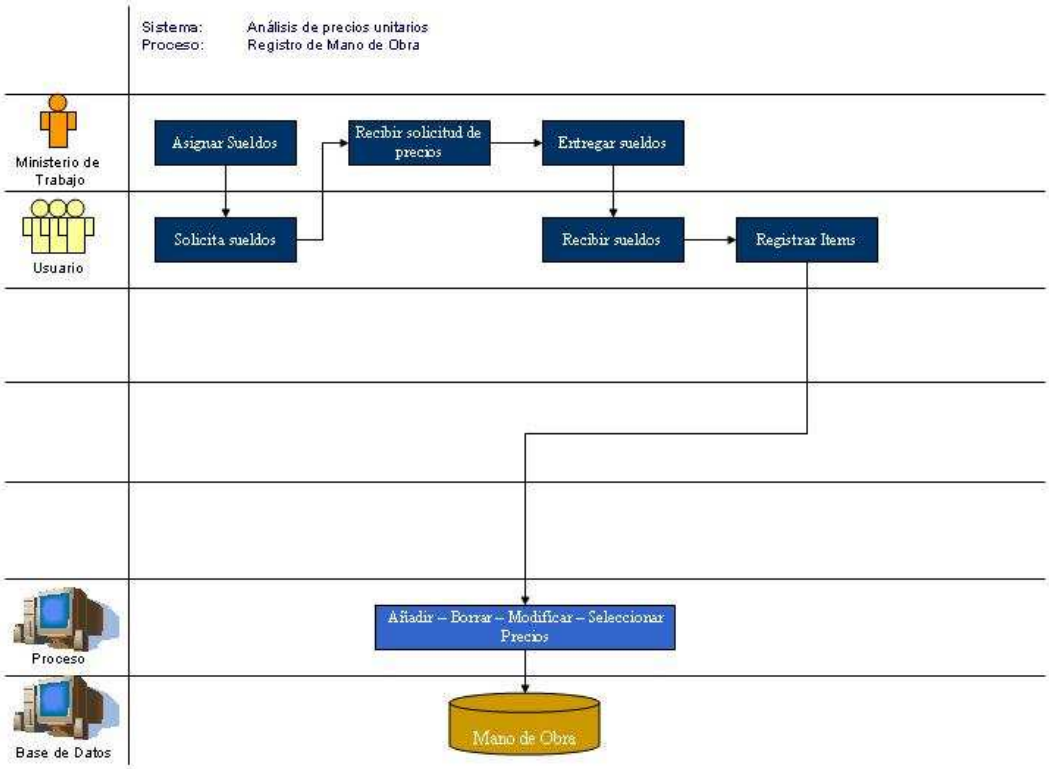


Figura A2.3: Flujo de procesos – Registro de mano de obra
Autor: Edison Hernández

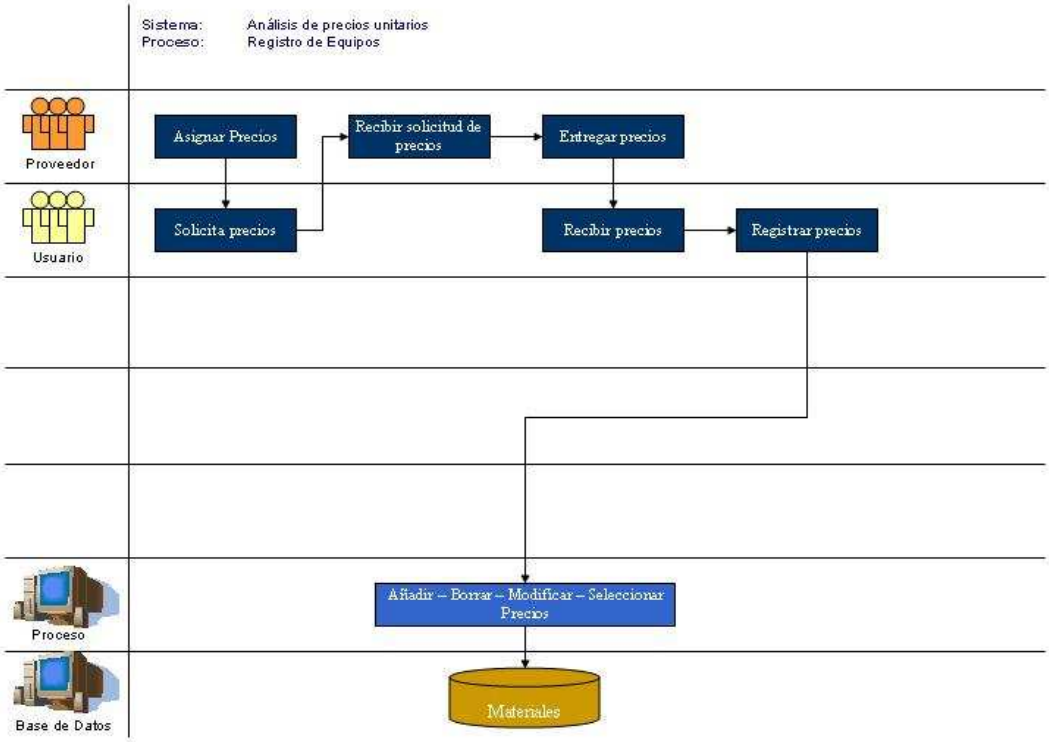


Figura A2.4: Flujo de procesos – Registro de equipos
Autor: Edison Hernández

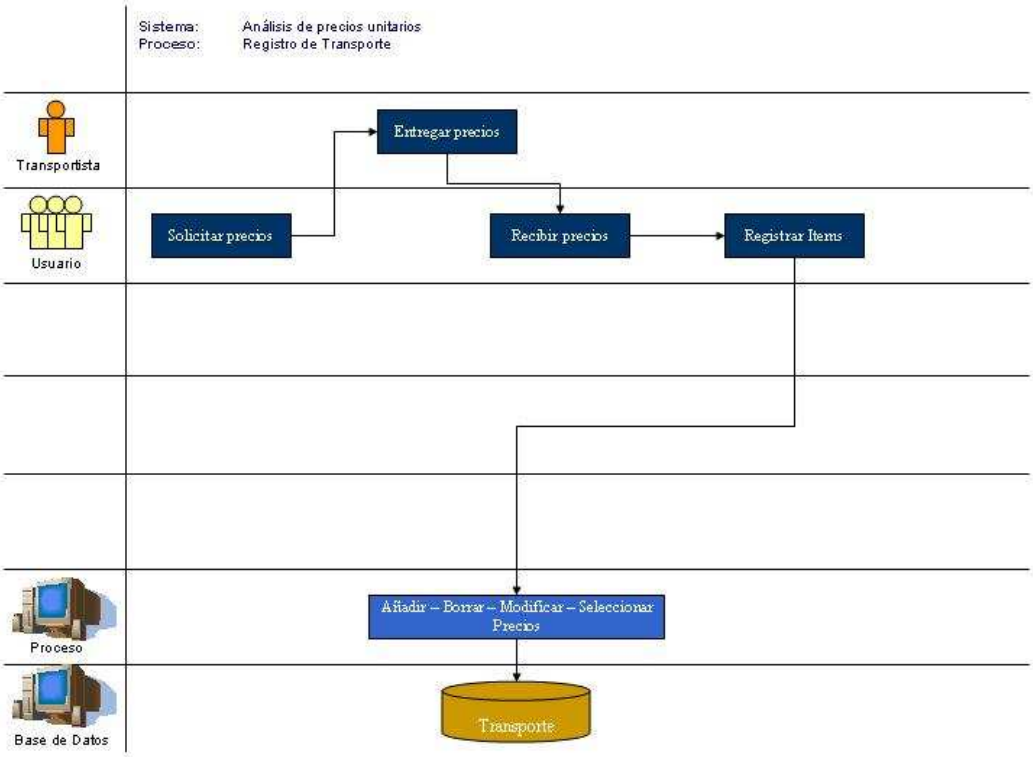


Figura A2.5: Flujo de procesos – Registro de transporte
Autor: Edison Hernández

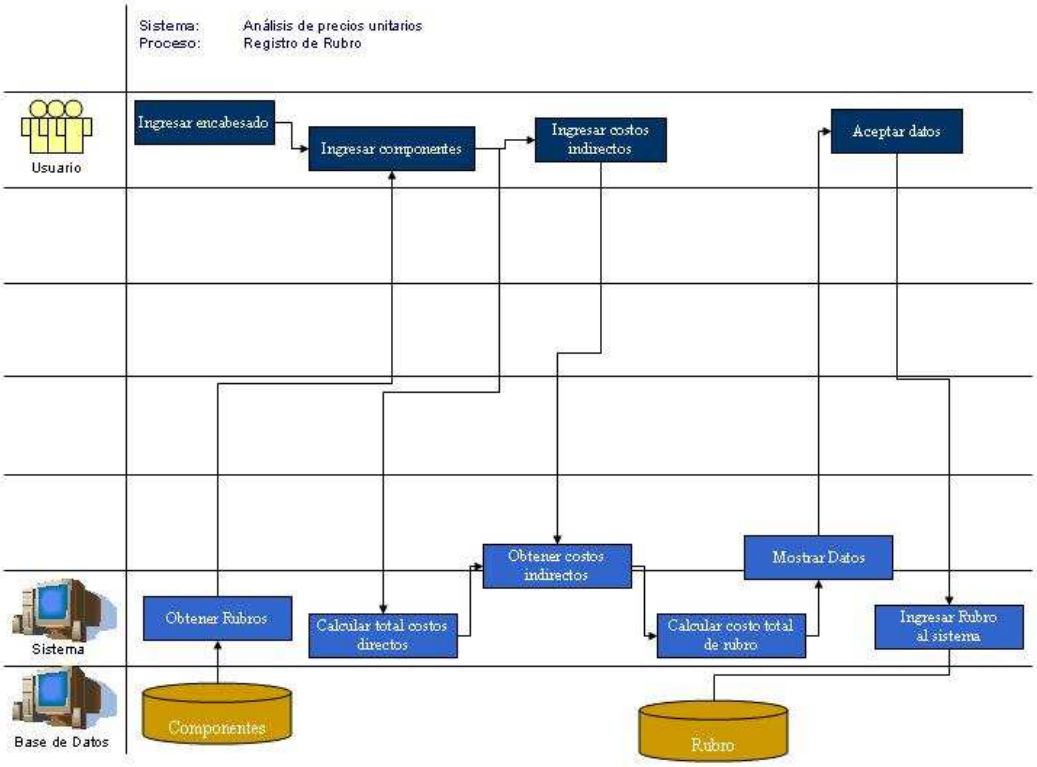


Figura A2.6: Flujo de procesos – Creación de rubros
Autor: Edison Hernández

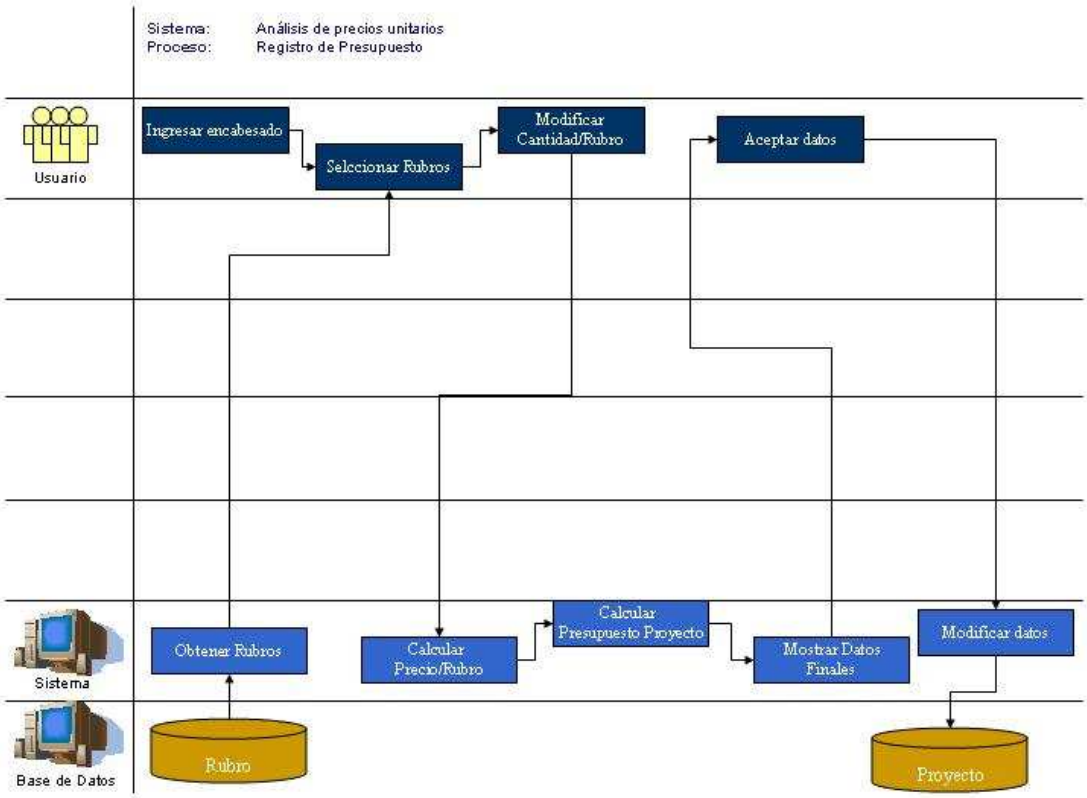


Figura A2.7: Flujo de procesos – Creación de presupuestos
Autor: Edison Hernández

ANEXO 3: INTERFACES DE USUARIO

Figura A3.1: Interface de usuarios – Ingreso al sistema
Autor: Edison Hernández

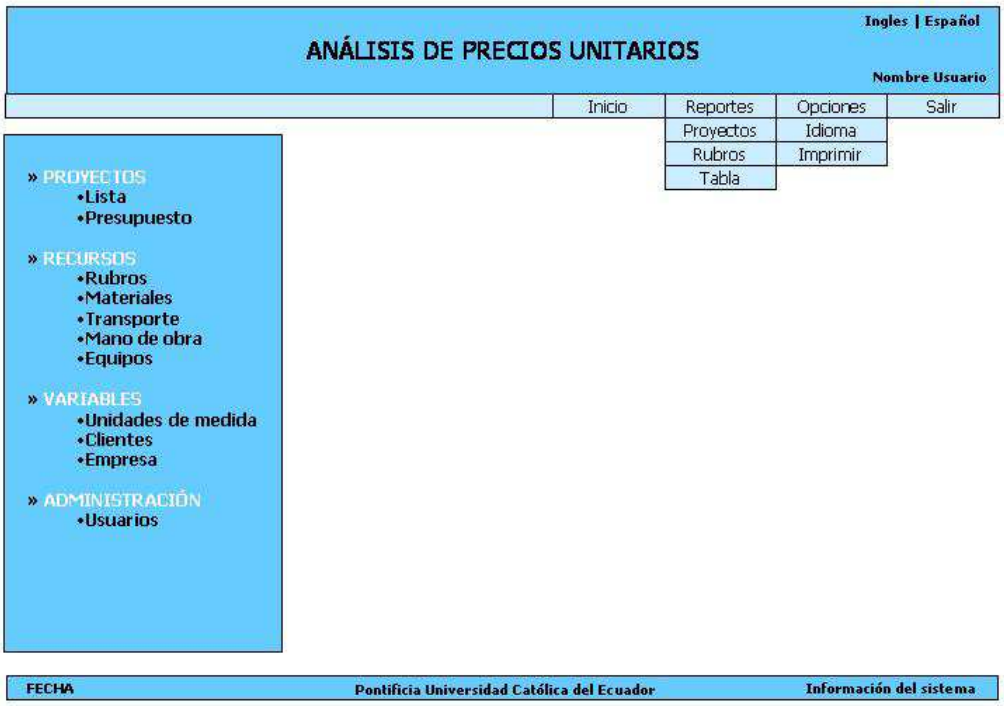


Figura A3.2: Interface de usuarios – Estructura básica del sistema
Autor: Edison Hernández

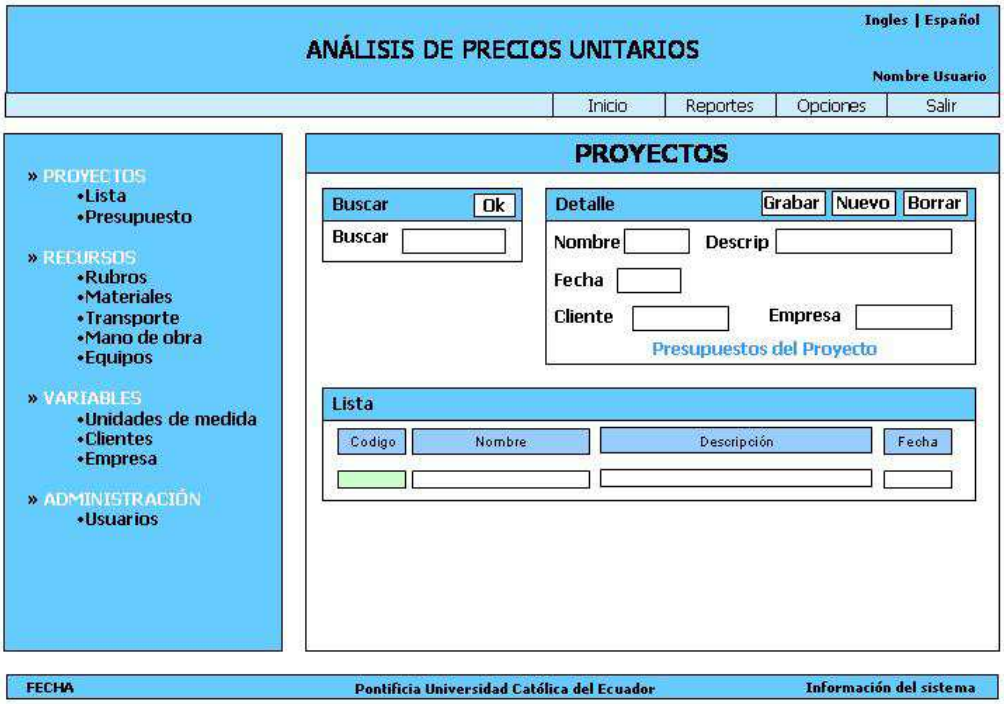


Figura A3.3: Interface de usuarios – Listado de proyectos
Autor: Edison Hernández

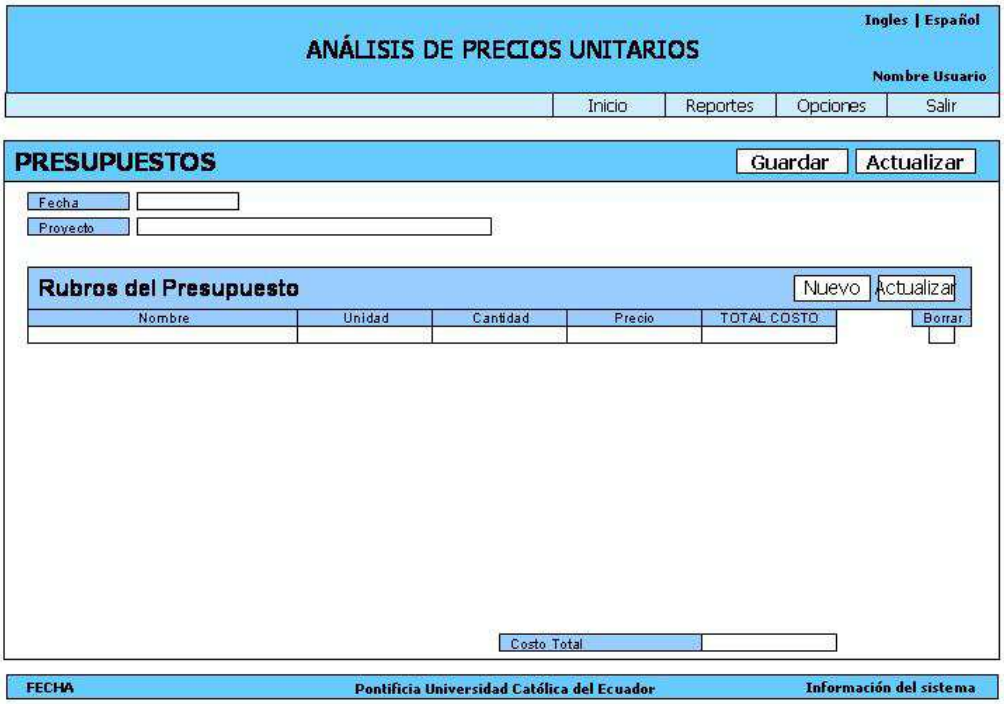


Figura A3.4: Interface de usuarios – Presupuestos
Autor: Edison Hernández

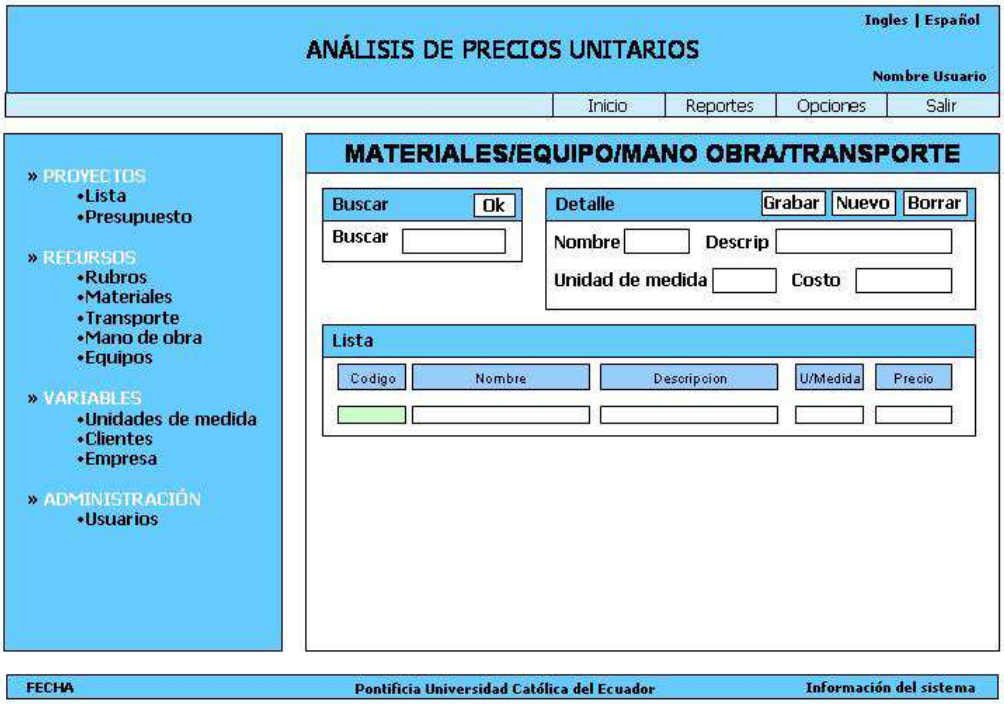


Figura A3.5: Interface de usuarios – Manejo de recursos
Autor: Edison Hernández

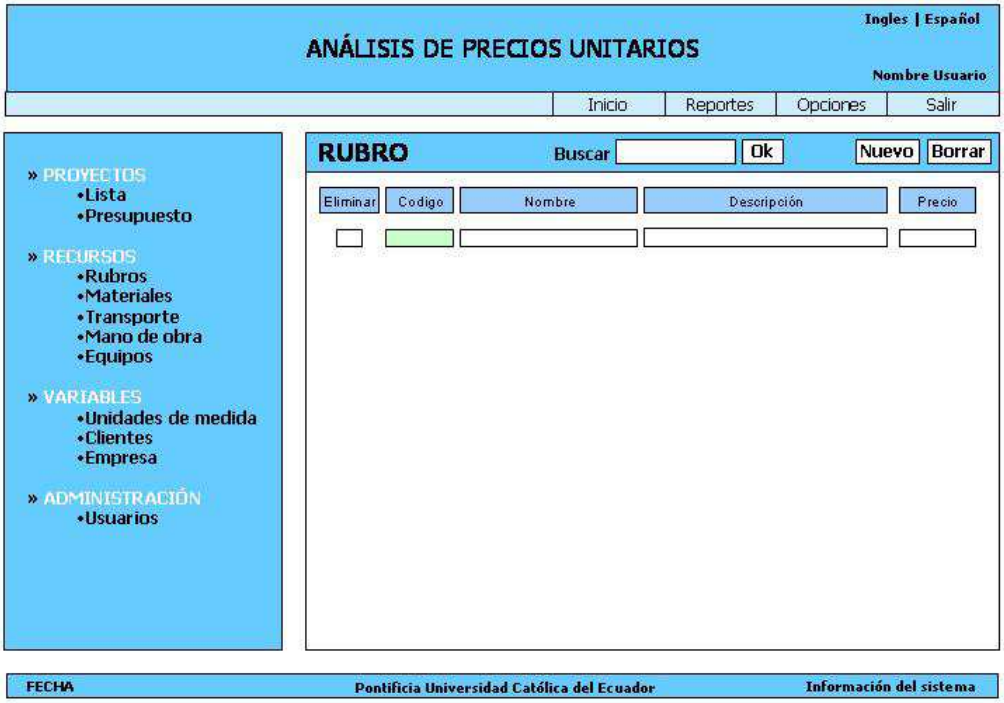


Figura A3.6: Interface de usuarios – Listado de Rubros
Autor: Edison Hernández

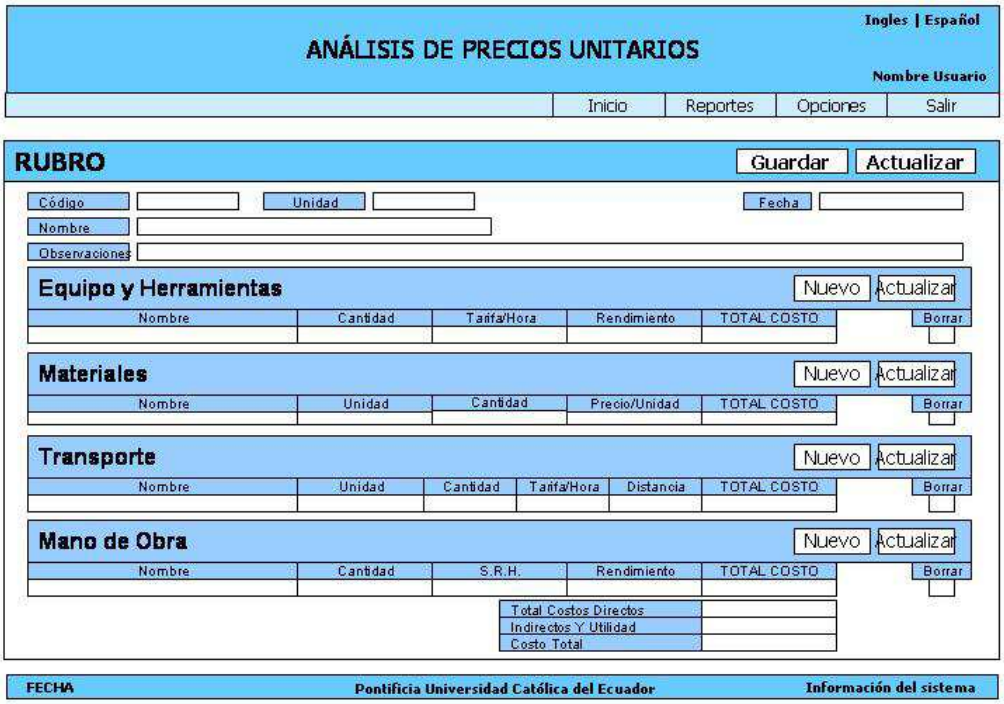


Figura A3.7: Interface de usuarios – Creación de rubros
Autor: Edison Hernández

RequestinsPage

BIBLIOGRAFÍA

- **Introducción**

Lindholm, Tim; Yellin, Frank. 1999. *The JavaTM Virtual Machine Specification*. Disponible en: <http://java.sun.com/docs/books/vmspec/2nd-edition/html/Introduction.doc.html>. (Teoría sobre la Maquina Virtual de Java)

Salvatierra, Marlo. 1997. *Publicidad en Internet*. Disponible en: <http://www.monografias.com/trabajos7/puin/puin.shtml>

Venegas, J. 1999. *Herramientas de Creación Web*. Universidad de Valladolid. Disponible en: <http://www.infor.uva.es/~jvegas/cursos/buendia/pordocente/node10.html>

Venegas, J. 1999. *Introducción a las Aplicaciones Web*. Universidad de Valladolid. Disponible en: <http://www.infor.uva.es/~jvegas/cursos/buendia/pordocente/node11.html>. (Conceptos básicos de html)

Venegas, J. 1999. *Desarrollo de Aplicaciones Web*. Universidad de Valladolid. Disponible en: <http://www.infor.uva.es/~jvegas/cursos/buendia/pordocente/node17.html>. (Conceptos sobre Arquitectura y Servidores web.)

Mendez, Justo. 1997. *Lenguajes de Programación*. Disponible en: <http://www.monografias.com/trabajos/lengprog/lengprog.shtml>. (Conceptos básicos de lenguaje Java>

Nango Quintana, Carlos Ernesto. *Chat´s en el lenguaje Java*. Disponible en: <http://www.monografias.com/trabajos12/chtjava/chtjava.shtml>. (Conceptos Básicos De Java, Cliente/Servidor)

MySQL AB. 2004. *MySQL Licensing Policy*. Disponible en <http://www.mysql.com/company/legal/licensing/>. (Políticas de licenciamiento de MySQL)

Sun Microsystems. *Java Frequently Asked Questions*. Disponible en: <http://java.sun.com/j2se/faq.html#Licensing>. (Licenciamiento de JDK)

Galáz, Solange. *Ingeniería de software*. Disponible en: <http://www.monografias.com/trabajos5/inso/inso.shtml>. (Etapas de la ingeniera de software para desarrollo de proyectos)

Torres Lecuanda, Araceli. *Metodologías modernas de desarrollo de Sistemas de Información*. Disponible en: <http://www.monografias.com/trabajos12/docmento/docmento.shtml>

- **J2EE**

Armstrong, Eric; Ball, Jennifer; Bodoff, Stephanie; Bode Carson, Debbie; Evans, Ian; Green, Dale; Haase, Kim; Jendroc, Eric. 1996. *The J2EE 1.4 Tutorial*. Disponible en: <http://java.sun.com/j2ee/1.4/docs/tutorial/doc/>

Sun Microsystems; traducido por Juan Antonio Palos. *Introducción a J2EE*. Disponible en: <http://www.programacion.com/java/tutorial/j2ee/>

Sun Microsystems. 2003. *The J2EE Tutorial (A beginner's guide to developing enterprise applications)*. Disponible en: <http://java.sun.com/j2ee/tutorial/1_3-fcs/>

Sun Developer Network Community. *White Paper - JavaServer Pages Technology - White Paper*. Disponible en: <http://java.sun.com/products/jsp/whitepaper.html>. (Conceptos básicos JSP)

Sun Developer Network Community. *White Paper - The Java Servlet API White Paper*. Disponible en: <http://java.sun.com/products/servlet/whitepaper.html>. (Conceptos básicos Servlets)

Sun Developer Network Community. *White Paper - JavaServer Pages Technology - JavaServer Pages Comparing Methods for Server-Side Dynamic Content White Paper*. Disponible en: <http://java.sun.com/products/jsp/jspservlet.html>

Sun Developer Network Communit. *JDBC API Overview*. Disponible en: <http://java.sun.com/products/jdbc/overview.html>. (Conceptos JDBC)

Sun Developer Network Community. *White Paper - JavaServer Pages[tm] Technology - JavaServer Pages Servlet Developer White Paper*. Disponible en: <http://java.sun.com/products/jsp/jsp-developerwp.html>. (Introduction to JavaServer Pages)

Sun Developer Network Community. *White Paper - JavaServer Pages[tm] Technology - JavaServer Pages White Paper*. Disponible en: <http://java.sun.com/products/jsp/jspguide-wp.html>. (JSP vs. CGI)

Díaz, Moisés Daniel. 2003. *Diseño de aplicaciones Internet usando los Patrones de diseño J2EE*. Disponible en: <http://www.programacion.com/java/articulo/corej2ee_patterns/>

Molpeceres Touris, Alberto; Pérez Mariñán, Martín. Diciembre, 2002. Arquitectura empresarial y software libre, J2EE. Disponible en: <http://www.javahispano.org/articles.article.action?id=70>. (J2EE – Herramientas gratuitas)

Sheil, Humphrey; Monteiro, Michael. Junio, 2002. *Rumble in the jungle: J2EE versus .Net, Part 1*. Disponible en: <http://www.javaworld.com/javaworld/jw-06-2002/jw-0628-j2eevsnet.html>. (Conceptos J2EE).

Fung, Kam Hay. Febrero, 2003. *Standard for JSP pages*. Disponible en: <http://developer.java.sun.com/developer/technicalArticles/javaserverpages/code_convention/>. (Convenciones de Programación Java para Páginas JSP)

Alarcón Mery, Constantino Samuel. Agosto, 2004. *Aplicación de Patrones J2EE en un Caso de Estudio*. Disponible en: <http://www.programacion.com/java/articulo/inukisoft/>

Sun Microsystems. *Developing Enterprise Applications Using the J2EE Platform*. Disponible en: <http://java.sun.com/developer/onlineTraining/J2EE/Intro2/j2ee.html>

Sun Developer Network Communit. *JavaServer Faces Technology Overview*. Disponible en: <http://java.sun.com/j2ee/javaserverfaces/overview.html>

Sun Developer Network Communit. J2EE Connector Architecture. Disponible en: <http://java.sun.com/j2ee/white/connector.html>

Sheil, Humphrey. Diciembre, 2001. *To EJB, or not to EJB?*. Disponible en: <http://www.javaworld.com/javaworld/jw-12-2001/jw-1207-yesnoejb.html>

Sun Microsystems; traducido por Juan Antonio Palos Título. Diciembre, 2003. *Catálogo de Patrones de Diseño J2EE. Y II: Capas de Negocio y de Integración*. Disponible en: <http://www.programacion.net/java/tutorial/patrones2/.>

Chamorro Villar, Ricardo. Noviembre, 2002. *Análisis comparativo entre Microsoft® .NET y Sun® J2EE*. Disponible en: <http://www.ciberteca.net/articulos/programacion/net/>

Zhong Jian. Septiembre, 2001. *Step into the J2EE architecture and process*. Disponible en: <http://www.javaworld.com/javaworld/jw-09-2001/jw-0928-rup.html>.

Rodriguez Lasterra, Enrique. Julio, 2002. *Introducción a Enterprise Java Beans*. Disponible en: <http://www.javahispano.org/articles.article.action?id=63>. (Enterprise Java Beans)

Gutiérrez, Gustavo A. *J2EE. Una plataforma para el cómputo empresarial*. Disponible en: <http://www.enterate.unam.mx/Articulos/dos/junio/j2ee.htm>

Sun Microsystems; traducido por Juan Antonio Palos. Junio, 2001. *Beans (Básico)*. Disponible en: <http://www.programacion.com/java/tutorial/beans/>

Sun Microsytems; traducido por Juan Antonio Palos. Mayo, 2001. *JavaBeans Enterprise*. Disponible en: <http://www.programacion.com/java/tutorial/javabeans/>

Sun Microsytems; traducido por Juan Antonio Palos. Mayo 2001. *Servlets y JSP*. Disponible en: <http://www.programacion.com/java/tutorial/servlets_jsp/>

Sun Microsytems; traducido por Juan Antonio Palos. Enero, 2003. *Introducción a los Servicios Web en Java*. Disponible en: <http://www.programacion.com/java/tutorial/servic_web/>

Abián, Miguel Ángel. Mayo, 2002. *J2ee Y .Net: La Rivalidad Permanente*. Disponible en: <http://www.javahispano.org/articles.article.action?id=55>

- **IDE**

Sun Microsystems. Noviembre, 2003. *Sun Java System Studio Standard 5 update 1Data Sheet*. Disponible en: <http://wwws.sun.com/software/sundev/jde/datasheet.html>

Storkel, Scout; traducido por Juan Antonio Palos. Septiembre, 2004. *Eclipse -- I -- Historia y Toma de Contacto*. Disponible en: <http://www.programacion.net/java/articulo/jap_eclip_1/>

Abián, Miguel Ángel. Febrero, 2003. *EL ARCHIPIÉLAGO ECLIPSE*. Disponible en: <http://www.javahispano.org/articles.article.action?id=75>

PC Magazine. Mayo, 2003. *Sun ONE Studio*. Disponible en: <http://www.pcmag.com/article2/0,1759,1191823,00.asp>

Sun Microsystems. *Sun Java Studio Standard 5 update 1 Features & Benefits*. Disponible en: <http://wwws.sun.com/software/sundev/jde/features/index.html>

Stephens, Matt. Abril, 2004. *First Look: Sun Java Studio Creator (Early Access)*. Disponible en: <http://www.softwarereality.com/reviews/creator_ea.jsp>

- **Struts**

The Apache Software Foundation. *Documentación Struts*. Disponible en: <http://struts.apache.org/userGuide/index.html>

The Apache Software Foundation; traducido por Juan Antonio Palos. Marzo, 2002. *El API Struts*. Disponible en: <http://www.programacion.com/java/tutorial/struts/1/>. (Conceptos de Struts y MVC)

Antoniucci, Javier. Abril, 2002. *Manual Básico de Struts*. Disponible en: <http://www.programacion.com/java/tutorial/joa_struts>. (Introducción a Struts)

- **Salmon SOFIA**

Salmon LLC. *Salmon LLC – SOFIA (Salmon Open Framework for Internet Applications)*. Disponible en: <http://www.salmonllc.com/website/Jsp/vanity/Sofia.jsp>

Dubinsky Dan. *Project SOFIA Forum*. Disponible en: <http://sourceforge.net/projects/salmon>

Santiago, Carlos. Marzo, 2004. *How does your framework compare to SOFIA*. Disponible en: <http://mail-archive.objectweb.org/barracuda/2004-03/msg00021.html>. (Introducción a SOFIA)

Dueñas, J. C. *Modelo-Vista-Controlador*. Disponible en: <http://polaris.dit.upm.es/~jcduenas/patrones/Modelo.htm>. (Introducción y contexto de MVC)

Sun Microsystems. 2002. *Designing Enterprise Applications with the J2EETM Platform, Second Edition*. Disponible en: <http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/titlepage.html>. Capitulo 4.3, 4.4. (Web-Tier Application Structure, Web-Tier Application Framework Design)

Simal Gándara, Juan. *Desarrollo de Aplicaciones con tecnología Internet*. Disponible en: <http://www.soluziona.es/htdocs/areas/consultoria/interes/articulos/tecnologia_internet.shtml>. (Arquitectura de las aplicaciones en tres capas, MVC)

Manzanedo del Campo, Miguel Ángel; García Peñalvo, Francisco; Cruzado Nuño, José Ignacio. *Guía de Iniciación al Lenguaje JAVA*. Capitulo IV. Disponible en: <http://pisuerga.inf.ubu.es/lsi/Invest/Java/Tuto/IV_3.htm>. (Conceptos de SWING)

- **Análisis de Precios Unitarios**

López Aguilar, Juan José. Noviembre, 2000. *Análisis de Precios Unitarios*. Disponible en: <http://www.monografias.com/trabajos6/anpre/anpre.shtml>. (Conceptos dentro del análisis de precios)

Garcia Carrasquero, Carlos. Noviembre, 2002. *Ajuste y reconsideración de precios en la construcción*. Disponible en: <http://www.monografias.com/trabajos11/ajus/ajus.shtml>. (Conceptos básicos de análisis de precios)

Murillo Rountree, Gabriel. 1989. *Administración de Empresas Constructoras*. 1ra edición .Guayaquil, Ecuador.

Merino, Wilfredo. 1979. *Manual de composición de costos unitarios para la construcción de carreteras*. Editorial Politécnica.