ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

Eguana Reports

Servidor de Reportes en Tecnología Java y XML

TESIS DE GRADO

Previa a la obtención del Título de:

Ingeniero en Computación Especialización Sistemas Tecnológicos

Ingeniero en Computación Especialización Sistemas Tecnológicos

Ingeniero en Computación Especialización Sistemas de Información

Presentada por:

Roy Stalin Cox Sosa

David Fernando Pérez Mawyín

José Xavier Pérez Sigüenza

GUAYAQUIL – ECUADOR

2009

DEDICATORIA

A nuestros padres, esposas, familiares y amigos.

TRIBUNAL DE GRADUACIÓN

MSc. Jorge Aragundi	Ing. Luis Muñoz
SUB-DECANO DE LA FIEC	DIRECTOR DE TÓPICO
MBA. Ana Tapia Rosero	MSc. Carmen Vaca Ruiz
MIEMBRO PRINCIPAL	MIEMBRO PRINCIPAL

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este Proyecto nos corresponde exclusivamente; y el patrimonio intelectual de la misma, a la Escuela Superior Politécnica del Litoral"

(Reglamento de exámenes y títulos profesionales de la ESPOL)

Roy Stalin Cox Sosa

David Fernando Pérez Mawyín

José Xavier Pérez Sigüenza

RESUMEN

Eguana Reports es un sistema de reportes desarrollado por los estudiantes del Tópico de Graduación “Desarrollo de Aplicaciones Transaccionales con Java y XML”.

El sistema demuestra el uso extenso de tecnología J2EE (Java 2 Enterprise Edition). Bajo esta plataforma se puede encontrar, dentro del proyecto, varias herramientas de código abierto integradas para alcanzar un objetivo. Asimismo, este documento puede constituir una guía práctica para aquellos desarrolladores interesados en J2EE.

Eguana Reports es una aplicación independiente. Puede integrarse con cualquier otra aplicación que utilice una fuente de datos y generar reportes a partir de estos.

ABREVIATURAS UTILIZADAS

AJAX:Asynchronous JavaScript And XML. JavaScript y XML Asíncrono.

API: Application Programming Interface. Interfaz de Programación de Aplicaciones.

CSV: Comma Separated Values. Valores Separados por Coma.

DAO: Data Access Object. Objeto de Acceso a Datos.

DHTML: Dynamic HTML. HTML Dinámico.

DOC: Extensión para archivos de Microsoft Word.

DOM: Document Object Model. Model de Objeto de Documento.

DTD: Document Type Definition. Definición de Tipo de Documento.

EIS: Enterprise Information System. Sistema de Información Empresarial.

EJB: Enterprise JavaBeans.

GWT: Google Web Toolkit. Kit de Herramientas Web de Google.

HTML: HyperText Markup Language.

HTTP: HyperText Transfer Protocol. Protocolo de Transferencia de Hipertexto.

HQL: Hibernate Query Language. Lenguaje de Consulta de Hibernate.

IBM: International Business Machines.

IDE: Integrated Development Environment. Ambiente Integrado de Desarrollo.

J2EE: Java 2 Enterprise Edition

J2SDK: Java 2 SDK.

JDBC: Java DataBase Connectivity. Conectividad Java a Base de Datos.

JDO: Java Databinding Objects. Objetos Java para Enlace de Datos.

JNDI: Java Naming and Directory Interface. Interfaz Java para Nombres y Directorios.

JRE: Java RunTime Environment.

JRXML: JasperReports XML.

JSF: Java Server Faces.

JSP: JavaServer Pages.

JSTL: JSP Standard Tag Library. Librería de Tags Estándar para JSP.

JTA: Java Transaction API.

LGPL: Lesser General Public License. Licencia Pública General Reducida.

MVC: Model-View-Controler. Modelo-Vista-Controlador.

OC4J: Oracle Container 4 Java.

OSI: Open Source Initiative. Iniciativa de Código Abierto.

PDF: Portable Document Format. Formato de Documento Portable

POJO: Plain Old Java Object.

RTF: Rich Text Format. Formato de Texto Enriquecido.

SAX: Simple API for XML.

SDK: Software Development Kit. Kit para Desarrollo de Software.

SOFIA: Salmon Open Framework for Internet Applications.

SQL: Structured Query Language. Lenguaje de Consulta Estructurado.

SSL: Secure Socket Layer. Capa de Conexión Segura.

UML: Unified Modeling Language. Lenguaje Unificado de Modelamiento.

WML: Wireless Markup Language.

XLS: Extensión para archivos de Microsoft Excel.

XML: Extensible Markup Language.

XSD: XML Schema Definition. Definición de Esquema XML.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El objetivo de este trabajo es demostrar la capacidad de la plataforma J2EE y el movimiento Código Abierto (Open Source, en inglés), además de la integración de cada uno de los componentes utilizados. Este proyecto fue desarrollado completamente con herramientas y plataformas de código abierto.

Eguana Reports, como módulo, es la solución a la necesidad de reportes del sistema de E-Guana. E-Guana es una iniciativa de implementación de un sistema E-Procurement. A pesar de esto, Eguana Reports no depende de los otros módulos para su funcionamiento. Es independiente y puede integrarse sin problema a otro sistema que incluya una fuente de datos para generar reportes.

La organización de este documento se detalla a continuación:

El primer capítulo, "Justificación y Objetivos", explica todos los conceptos y planteamientos del proyecto, además de justificar las herramientas y plataformas a utilizar.

El segundo capítulo, "Análisis", detalla las herramientas, versiones y su interacción, además del análisis de las funcionalidades del proyecto.

El tercer capítulo, “Diseño – Arquitectura del Servidor de Reportes”, muestra los conceptos de diseño y arquitectura MVC aplicado al proyecto, el diseño de la aplicación y sus módulos.

El cuarto capítulo, “Implementación”, explica detalles técnicos de instalación, configuración e implementación de los componentes principales del proyecto.

Por último, se presentan las conclusiones y recomendaciones, bibliografía, glosario y anexos.

CAPÍTULO 1

1. Justificación y objetivos

Los reportes son indispensables, desde aplicaciones pequeñas a grandes, ya que permiten analizar información y tomar decisiones. En un mundo con un dramático crecimiento en tecnologías de la información, donde se desea altos niveles de competitividad, los reportes son herramientas muy necesarias para medir la situación de una empresa, planificar su rumbo y solucionar problemas.

El poder de la computación juega un papel crucial para producir reportes unificados. Este proyecto utiliza este poder y permite producir reportes que, tomando en cuenta la diversidad de información de una empresa, plasmen los diferentes puntos de vista de las personas involucradas.

1.1. Descripción del proyecto tecnológico

El proyecto se construye con tecnología de código abierto, el estándar J2EE y XML, para probar su integración y funcionalidad para iniciativas útiles y de necesidades reales en un ambiente empresarial.

La mayoría de sistemas tiene su propio módulo de reportes. Muchos son de alcance limitado o son difíciles de personalizar. Siendo así, la fuente de datos se vuelve heterogénea y hay que recurrir a distintos medios para obtener y consolidar la información de una empresa.

El proyecto Eguana Reports tiene como objetivo presentar una base tecnológica que permita, de manera dinámica, obtener reportes personalizables. Los reportes pueden ser diseñados por los usuarios, pudiendo elegir los campos a usar, el formato de salida o la fuente de datos. De esta manera, si se desea consultar información de diferentes aplicaciones, se tiene una fuente única de estos reportes.

Por otro lado, con los recursos asignados al desarrollo de reportes, se puede mejorar la calidad del código. Para lograrlo hay que dedicar cierta cantidad de recursos, como desarrolladores, tiempo o dinero. Los mismos recursos muchas veces permiten liberar un producto en menos tiempo.

Al tener que mantener una fuente única de acceso a la información se pueden liberar recursos y hacer más rentable a la organización por medio del proyecto Eguana Reports.

1.2. Objetivos del proyecto

De acuerdo con la necesidad de disminuir la dispersión de información y permitir a los usuarios de diferentes sistemas validar, revisar y deducir nueva información desde un punto central, se propone como solución el proyecto Eguana Reports.

Este proyecto tiene como objetivos:

- Utilizar herramientas de código abierto (open source), ya que son de fácil acceso y de bajo o cero costos de adquisición, lo que aminora los costos del proyecto y del mantenimiento de la tecnología.

- Crear reportes en formato PDF, HTML, XLS, CSV. Formatos comúnmente usados y de fácil exportación, en caso de ser necesario, a sistemas externos a la organización.

- Obtener reportes a partir de plantillas (formatos predefinidos). Estas plantillas son previamente grabadas en el servidor.

- Proveer un módulo para crear reportes personalizados.

- Tener un módulo propio para administrar usuarios, reportes y fuentes de datos. Esto nos permitirá organizar, restringir y delegar el acceso a los datos y su presentación.

- Definir un esquema de seguridad básica para el acceso a los reportes.

- Unificar la fuente de datos y de reportes dentro de una empresa.

- Proveer un método de integración con sistemas existentes.

- Permitir al equipo de desarrollo enfocar los recursos al sistema y no a módulos de reporte.

1.3. Código Abierto (Open Source)

Empezaremos con la propia definición del movimiento código abierto: Según la Iniciativa de Código Abierto (Open Source Initiative, en inglés)[1], “el código abierto es un método de desarrollo de software que aprovecha el poder de revisión por pares distribuidos y transparencia del proceso. La promesa del código abierto es: la mejor calidad, mayor fiabilidad, más flexibilidad, menor costo, y un fin a las restricciones de los vendedores de software, que depredan la capacidad de mejora.

[1] Fuente: Iniciativa de Código Abierto (Open Source Initiative, en inglés) http://www.opensource.org

La Iniciativa de Código Abierto (Open Source Initiative, OSI) es una corporación sin fines de lucro creada para educar sobre y abogar por los beneficios del código abierto y construir puentes entre los distintos grupos en la comunidad.

Una de nuestras actividades más importantes es ser un organismo de normalización, mantener de la definición de “Código Abierto” para el bien de la comunidad. La Iniciativa Código Abierto crea un nexo de ***confianza*** en torno a la cual los desarrolladores, usuarios, empresas y gobiernos pueden organizar una cooperación mutua.”

La definición es clara. Todos hemos tenido alguna vez la dificultad de obtener un programa específico por su alto costo, y tal vez en alguna ocasión hemos deseado poder tener acceso al código para mejorarlo. El principio que rige detrás del código abierto indica que estos problemas no existirán dentro de esta comunidad.

Hay miles de artículos disponibles en Internet en torno a “Código Abierto vs. Código Propietario”. Las principales diferencias se pueden resumir, a nuestro juicio, en la siguiente tabla:

Código Abierto	Código Propietario
No hay tasas de adquisición y mantenimiento del software (aunque algunos proveedores ofrecen acuerdos de mantenimiento, como en Red Hat Linux)	Hay costos de adquisición del software y de mantenimiento (comúnmente son suscripciones o pagos por actualizaciones)
Hay personal especializado de soporte, usualmente los propios creadores y colaboradores del software..	Hay personal de soporte listo y disponible, usualmente técnicos asignados por la empresa propietaria del software.
Usualmente el software necesita hardware poco costoso para ejecutarse.	Los requerimientos de hardware pueden ser muy costosos.
Las actualizaciones de un producto son compatibles con versiones anteriores del mismo producto.	Muchas veces se necesita instalar nuevas versiones del producto, así sea innecesario.

Tabla 1: Código Abierto vs. Código Propietario.
Fuente: Proyecto Eguana Reports

Se deben considerar otros factores:

- Ser parte de esta comunidad requiere una habilidad ganada por la experiencia en el uso y pruebas de software de código abierto. Esta habilidad es una base importante al momento de buscar y elegir el software de código abierto que cubra eficientemente las necesidades.

- Se puede decir, casi con certeza, que todo software de código abierto fomenta el libre uso, pero al mismo tiempo el autor o autores no se hacen responsables por daños y perjuicios causados al utilizar dicho software.

Un ejemplo evidente de lo que implica la comunidad de código abierto es SourceForge[2], el sitio web más grande para desarrollo de software de este tipo. Para inicios del 2009, SourceForge alberga más de 220.000 proyectos y 2 millones de usuarios registrados. Esto nos da una idea del vasto menú de aplicaciones disponibles, gente involucrada y lo valioso que resulta la experiencia en el desenvolvimiento dentro de esta comunidad.

Los proyectos de código abierto se distribuyen bajo licencia GNU LGPL (Licencia Pública General Reducida) (Lesser General Public License), creada por la Fundación de Software Libre. El objetivo de la licencia es garantizar la libertad de compartir y modificar el software, y por lo tanto que el software es libre para todos sus usuarios.

Por el mismo hecho de su apertura sin restricciones, y la cantidad de gente trabajando, el software de código abierto a veces puede tornarse en tema complejo.

Al final, la decisión correcta acerca de qué software utilizar, si código abierto o propietario, depende de la situación particular del usuario. Nosotros, dada la naturaleza de nuestro proyecto, tecnologías seleccionadas, objetivos deseados y bajos costos, utilizaremos código abierto. A continuación entraremos en detalle.

---

[2] SourceForge: http://www.sourceforge.net

1.4. Justificación de la tecnología J2EE

Plataforma Java Edición Empresarial (Java EE) (Java Platform Enterprise Edition, en inglés), conocida formalmente como J2EE, se construye sobre la sólida base de la Plataforma Java Edición Estándar (Java SE) (Java Platform Standard Edition). Es una plataforma ampliamente utilizada para el desarrollo de aplicaciones empresariales multicapas, y es considerada el estándar industrial para implementar arquitecturas empresariales orientadas a objetos.

Una plataforma describe la arquitectura de hardware y software y típicamente incluye arquitectura, lenguaje de programación, librerías e interfaces.

El nombre J2EE es usado hasta la versión Java EE 1.4. En versiones posteriores el término usado es Java EE. De aquí en adelante usaremos el término Java EE.

Java EE, además, constituye un conjunto de estándares, o colección de especificaciones y normas, para el desarrollo e implementación de aplicaciones distribuidas.

Informalmente, se considera a Java EE un estándar por el acuerdo que deben lograr los proveedores para considerar una aplicación compatible con

Java EE. La premisa de Java “desarrolla una vez y ejecuta en cualquier lugar” es uno de los motivos para utilizar esta plataforma: la *portabilidad*.

En este aspecto, y ya que las especificaciones deben ser aprobadas en consenso por comités de expertos, las aplicaciones tienen el potencial de ser *escalables* y de incluir otras características como *manejo de transacciones, concurrencia, seguridad y tolerancia a fallas*.

Java EE provee la infraestructura para soportar aplicaciones y un conjunto de Java API[3] (Application Programming Interface) para construirlas. Con las especificaciones y APIs los fabricantes pueden desarrollar su propia solución. La *variedad* de soluciones constituye otro factor para elegir esta plataforma. Por ejemplo, tenemos disponible en el mercado 9 o más servidores de aplicaciones, entre ellos WebSphere, WebLogic, JBoss y OC4J.

Cabe acotar que no necesariamente una aplicación Java EE implica que es de código abierto. El caso de OC4J es un ejemplo de un servidor de aplicaciones desarrollado por Oracle, sin la participación abierta de la comunidad. JBoss, en cambio, ha sido desarrollado por la comunidad de código abierto. Ambos cumplen las especificaciones Java EE.

---

[3] API significa Interfaz de Programación de Aplicaciones. Hablando de manera general, un API es un conjunto de clases dentro de una biblioteca que proveen una funcionalidad para ser utilizado por otro software.

Luego de mostrar los detalles más básicos de Java EE, daremos un breve resumen de las herramientas de trabajo que escogimos junto con las principales características.

Aplicaciones distribuidas multicapas

La plataforma Java EE utiliza el modelo de aplicación multicapas (multi-tier, en inglés) para aplicaciones empresariales. La lógica de la aplicación es dividida entre sus componentes de acuerdo a su función y la capa que representan, y a su vez, cada componente podría funcionar en máquinas diferentes.

A continuación mencionaremos de manera básica las capas de este modelo;

- Capa del cliente (client tier).- Los componentes se ejecutan en la máquina del cliente, por ejemplo páginas HTML dinámicas.

- Capa web (web tier).- Los componentes se ejecutan en la máquina del servidor Java EE. Las páginas JSP[4] se encuentran en esta capa.

- Capa de negocios (Business tier).- Los componentes se ejecutan en el servidor Java EE, contienen la lógica de negocio. Típicamente se hace referencia a Enterprise Beans.

---

[4] JavaServer Pages (JSP) es una tecnología Java desarrollada por Sun Microsystems que permite generar contenido dinámico para aplicaciones basadas en web.

- Capa de sistema de información (Enterprise Information System tier). Los componentes se ejecutan en el servidor de sistema de información. Una base de datos dispuesta en un servidor dedicado es el ejemplo más claro de esta capa.

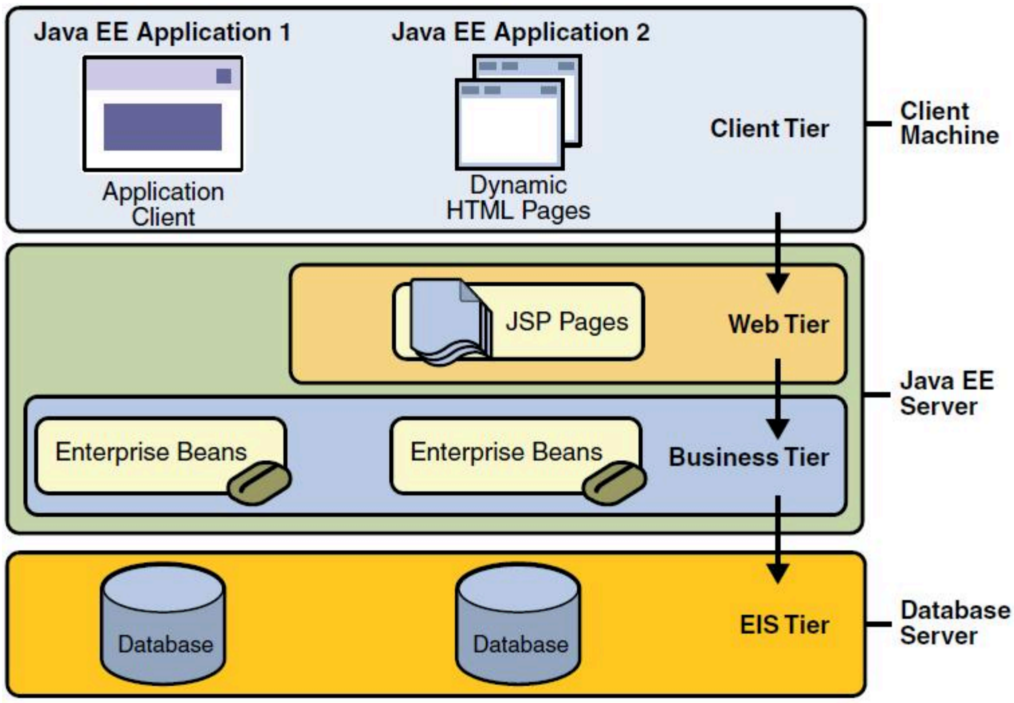


Figura 1: Modelo multicapas

Componentes Java EE

Las aplicaciones Java EE están formadas por componentes. Un componente Java EE es una unidad de software funcional que se ensambla a una aplicación Java EE con sus clases y archivos relacionados y que se comunica con otros componentes. La especificación Java EE define los siguientes componentes:

- Aplicaciones clientes y applets[5].- Componentes que se ejecutan en el cliente.
- Componentes de tecnología JSP (Java Server Pages) y Java Servlet.- Componentes Web que se ejecutan en el servidor.
- Componentes EJB (Enterprise JavaBeans).- Componentes de negocios que se ejecutan en el servidor.

Los componentes Java EE están escritos en lenguaje Java.

Un componente Java EE, para ensamblarse a una aplicación Java EE, debe cumplir con las especificaciones Java EE

Contenedores Java EE

La arquitectura Java EE hace que las aplicaciones Java EE sean fáciles de implementar, gracias a que se construye a partir de componentes e independencia de plataforma. Esta arquitectura abstrae al desarrollador de las complicaciones de implementar manejo de transacciones, concurrencia, multihilos y otros detalles de bajo nivel en aplicaciones distribuidas.

---

[5] Un applet es un componente de una aplicación que se ejecuta en el contexto de otro programa, no es independiente y debe ejecutarse en un contenedor que lo proporciona la aplicación anfitriona.

De aquí que un servidor Java EE provee los servicios necesarios a todo componente bajo el término de Contenedor Java EE. No es necesario que el desarrollador cree nuevos componentes, simplemente usa los que provee el servidor, y concentra sus esfuerzos en resolver problemas del negocio.

Tipos de contenedores

El despliegue de una aplicación Java EE se hace dentro de contenedores Java EE:

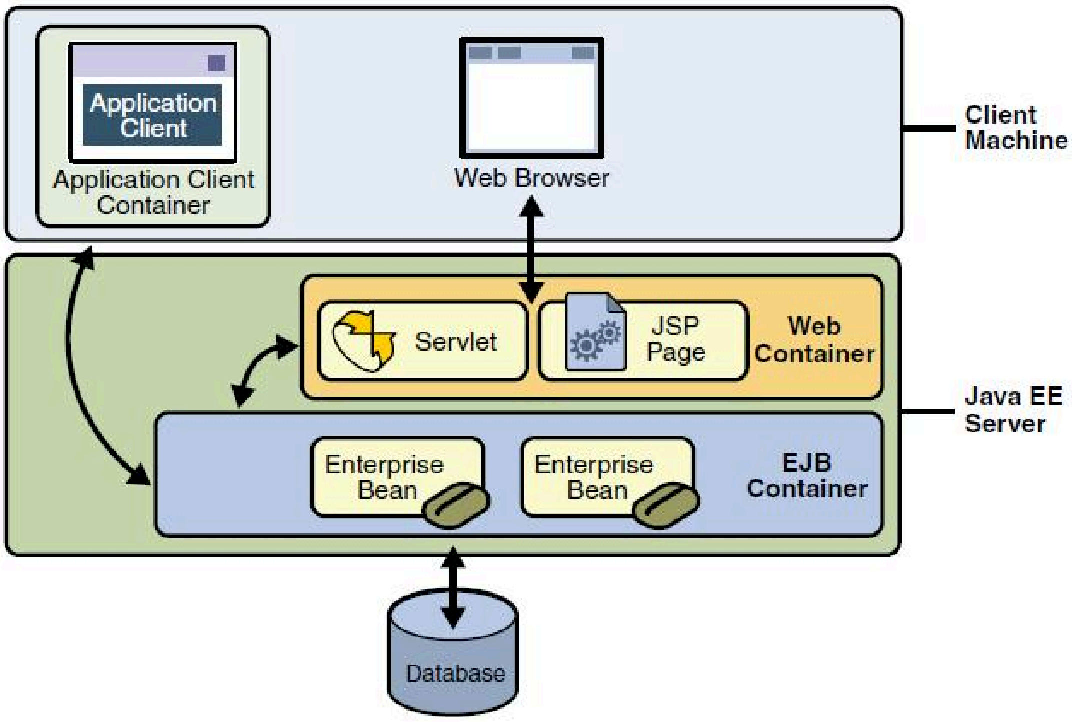


Figura 2: Servidor y Contenedores Java EE

- Servidor Java EE (Java EE Server).- Es la parte ejecutable de todo el producto Java EE, y provee los contenedores Web y EJB.

- Contenedor EJB (Enterprise Java Beans Container).- Administra la ejecución de beans[6] empresariales en las aplicaciones Java EE. Los EJBs y su contenedor se ejecutan en el servidor Java EE.

- Contenedor Web (Web Container).- Maneja la ejecución de páginas JSP y componentes servlets para aplicaciones J2EE. Estos componentes y su contenedor se ejecutan en el servidor J2EE.

- Contenedor de aplicación cliente (Application client container).- Administra la ejecución de componentes de la aplicación cliente. La aplicación y su contenedor se ejecutan en el cliente.

- Contenedor Applet (Applet container).- Administra la ejecución de applets. Consiste de un navegador web y plug-in[7] Java ejecutándose juntos en la máquina cliente.

1.4.1. JBoss – Servidor de Aplicaciones

JBoss Application Server es la implementación en código abierto de un conjunto de servicios Java EE, desarrollado por la comunidad y

[6] Bean, traducido literalmente como grano, es un término que define a un componente que cumple una función específica y que puede ser reutilizado por otros desarrolladores o aplicaciones.

considerado como uno de los servidores de aplicaciones más utilizado a nivel mundial.

Es una plataforma Java EE para desarrollo e implementación de aplicaciones Java EE, aplicaciones web, y provee una amplia gama de servicios y componentes Java EE junto con otros servicios empresariales como persistencia y caché.

JBoss, sobre la base de Java, es portable y se puede instalar en cualquier plataforma que soporte Java.

Tomcat – Contenedor web

Apache Tomcat es una implementación en código abierto para las tecnologías Java Servlets y Java Server Pages (JSP), desarrollada por la comunidad de Apache Software Foundation bajo licencia de Apache. Es un producto del ambiente colaborativo entre desarrolladores de todo el mundo.

Muchos consideran a Tomcat como un servidor HTTP escrito en Java que soporta Servlets y JSP. En tal caso su principal característica es la de contenedor web.

---

[7] Plug-in: Complemento a una aplicación para aportarle una función adicional. En este caso el plug-in Java adiciona funcionalidad Java al navegador web.

Originalmente es un producto de Sun, que ahora lo desarrolla Apache Software Foundation, y es considerado una referencia a implementación de Servlets y JSP antes que un servidor de producción. Sin embargo, su facilidad y versatilidad lo han hecho elegible para muchos ambientes en pequeñas, medianas y grandes empresas con procesos web críticos, obteniendo resultados exitosos.

Es importante destacar que JBoss tiene embebido Tomcat como su contenedor web.

Finalmente, Eguana Reports no requiere el uso de un contenedor EJB, por razones que veremos más adelante. Nuestras necesidades son cumplidas muy bien con el uso de Tomcat como nuestro contenedor web.

1.4.2. MySQL – Base de Datos

La base de datos MySQL es la base de datos de código abierto más popular del mundo. Es fácil de usar, es confiable y tiene excelente rendimiento. Es muy utilizada alrededor de todo el mundo en muchas aplicaciones web y otras aplicaciones críticas.

MySQL es simple. Lo utiliza Yahoo!, Nokia, YouTube y Google, por mencionar algunos ejemplos.

Se puede ejecutar en múltiples plataformas incluyendo Linux, Windows, OS/X, HP-UX, AIX, Netware. Muchas de las tareas de mantenimiento e integridad de datos las deja en las manos del desarrollador o del servidor de aplicaciones y se preocupa solamente en cumplir las tareas específicas de una base de datos. Esta filosofía de desarrollo ha permitido que MySQL sea una base datos de gran aceptación.

MySQL propone varias razones para escoger este producto:

- Escalable y flexible.- Multiplataforma y naturaleza de código abierto.

- Alto desempeño.- Se puede configurar para situaciones específicas, tanto para manejo de grandes volúmenes de información como para transacciones que requieren velocidad.

- Amplia disponibilidad.- Ofrece múltiples opciones para un espectro amplio de usuarios.

- Soporte transaccional robusto.- Integridad referencial reforzada y características de soporte de transacciones en todo nivel.

- Fortaleza para web y data Warehouse.- Dado por su motor de alto rendimiento con buena capacidad de inserción de datos.

- Protección de datos.- Provee mecanismos para autenticación y bloqueo de usuarios, acceso restringido a los datos y soporte SSL[8] para conexión segura.

- Desarrollo de aplicaciones comprensible.- Soporte para cada necesidad de desarrollo.

- Facilidad de administración.

- Código abierto.

- Bajo costo.

MySQL es una base de datos relacional. Para integrarla con la aplicación Java EE se utiliza una herramienta de mapeo objeto-relacional, que en nuestro caso es Hibernate.

1.4.3. Struts – Framework para implementar arquitectura MVC

Struts es un framework[9] para crear aplicaciones Java para la web basado en arquitectura MVC[10]. Está diseñado para aumentar la productividad del ciclo de desarrollo, desde la creación hasta la implementación.

---

[8] SSL (Secure Socket Layer), Capa de Conexión Segura.
[9] Un marco o conjunto de herramientas, programas que se especializan en proveer una funcionalidad específica.
[10] Model-View-Controller, en castellano Modelo-Vista-Controlador.

Struts se desarrollaba como parte del proyecto Jakarta de Apache Software Foundation, pero actualmente se desarrolla independientemente bajo el nombre Jakarta Struts.

Se basa en tecnología estándar (XML, JSP, Servlets, HTML, entre otros), y una de sus principales características es que permite separar la lógica de presentación de la lógica de negocios. Struts enfatiza el uso del Modelo 2 en la arquitectura de aplicaciones.

Para implementar la arquitectura MVC, Struts provee su propio Controlador y se integra con otras tecnologías para proporcionar la Vista y el Modelo. Para el modelo, Struts puede interactuar con tecnologías estándares de acceso de datos y tecnologías desarrolladas por terceros (como Hibernate). Para la Vista, Struts trabaja bien con JSP y otros sistemas de presentación.

Es una gran ayuda en el desarrollo de aplicaciones web. Entre otras características tiene un esquema MVC ya definido, soporta internacionalización y procesamiento de formas.

1.4.4. Hibernate – Herramienta de Mapeo Objeto-Relacional

Hibernate es un servicio de query y persistencia objeto-relacional. Como tal, permite al desarrollador la creación de clases persistentes utilizando la programación orientada a objetos para luego integrarlas con el mundo de las fuentes de datos relacionales.

Para integrar las clases con una fuente de datos relacional se utilizan los archivos de mapeo, que no son otra cosa que archivos donde se detalla la relación entre los atributos de un objeto con los registros de la fuente de datos, y de las clases con las respectivas entidades en la fuente de datos (comúnmente con tablas).

Hibernate convierte los datos entre los tipos utilizados por Java y los definidos por SQL. También permite hacer consultas usando propio lenguaje de consultas HQL, o con Criteria[11] orientada a objetos o con SQL nativo.

Hibernate es un proyecto de código abierto y actualmente es un componente crítico de JBoss en los sistemas empresariales. Se distribuye bajo la licencia LGPL.

1.4.5. Jasper Reports – Herramienta de Reportes

Jasper Reports es una librería de clases Java en código abierto diseñada para ayudar a los desarrolladores en la tarea de añadir capacidades de generación de reportes a las aplicaciones Java,

[11] Criteria es un API simplificado para consulta que retorna objetos según ciertos criterios.

proporcionando un API para facilitar la generación de reportes desde cualquier tipo de aplicación Java.

Es un potente motor para la generación de reportes que tiene la habilidad de entregar buen contenido a la pantalla, impresora o archivos PDF, HTML, XLS, CSV y XML. Puede decirse que es el motor de código abierto para reporte más utilizado en el mercado. El propósito principal es crear reportes listos para la entrega/impresión de una manera simple, flexible y dinámica.

Jasper Reports organiza datos extraídos de una base de datos de acuerdo a un diseño de reporte (definido en XML y luego compilado). El diseño es un archivo de extensión JRXML.

El diseño se compila y se guarda como un objeto de reporte, para luego ser llenado con datos. El diseño compilado es un archivo con extensión .JASPER, y es el que se carga en la aplicación y se llena con datos en tiempo real para producir el reporte final.

El resultado es un objeto que representa un documento listo para entregar/imprimir. Este objeto puede ser una página HTML, un archivo Excel, un PDF, o cualquier otro.

Jasper Reports fue al inicio un proyecto de un sólo hombre, creado para satisfacer sus necesidades de un motor de reportes de bajo costo. Hoy es una herramienta muy popular y, tal vez, es la herramienta Java para reportes más utilizada.

1.4.6. Castor XML – Framework de Mapeo Objeto-XML

Castor XML es lo que se denomina un “XML databinding framework”, y se puede definir como un conjunto de clases que proveen la funcionalidad necesaria para el proceso de representar la información en un documento XML como un objeto en memoria.

En pocas palabras, convertir un documento XML en su objeto Java equivalente y viceversa. En este contexto existen dos términos a saber:

- Marshal.- Consiste en convertir un Objeto a XML (un flujo de datos, secuencia de bytes)

- Unmarshal.- Consiste en convertir XML a Objeto.

Castor XML puede convertir casi cualquier objeto Java parecido a un bean[12] en XML y viceversa. En este proceso de conversión Castor utiliza como guía descriptores de clases y atributos para saber cómo debe hacer marshal y unmarshal entre Objeto y XML.

---

[12] Nos referimos a un simple bean para modelar datos, que contiene atributos con sus respectivos get() y set().

A diferencia de los otros APIs XML importantes, DOM (Document Object Model) y SAX (Simple API for XML), Castor permite lidiar con la información contenida en el documento XML sin tener que lidiar directamente con la estructura XML (nodos, root, sibling, childs, y los demás).

Ya que Jasper Reports guarda sus diseños con una estructura XML, Castor permitirá convertir ese diseño en objetos para trabajar en la creación dinámica de reportes dentro de nuestra aplicación Java EE.

1.4.7. Servlets

Los Servlets son módulos de código escrito en Java que añaden funcionalidad a un servidor Web. Extiende las capacidades del servidor de aplicaciones que son accedidas mediante el modelo de programación petición-respuesta (request-response). Fueron diseñados para aceptar peticiones de un cliente y generar los mensajes de respuesta correspondiente. Los servlets constituyen otro componente de una aplicación Java EE. El contenedor web (que contiene al servlet) es responsable por el mantenimiento del ciclo de vida del este.

Este es el esquema de funcionamiento de un servlet:

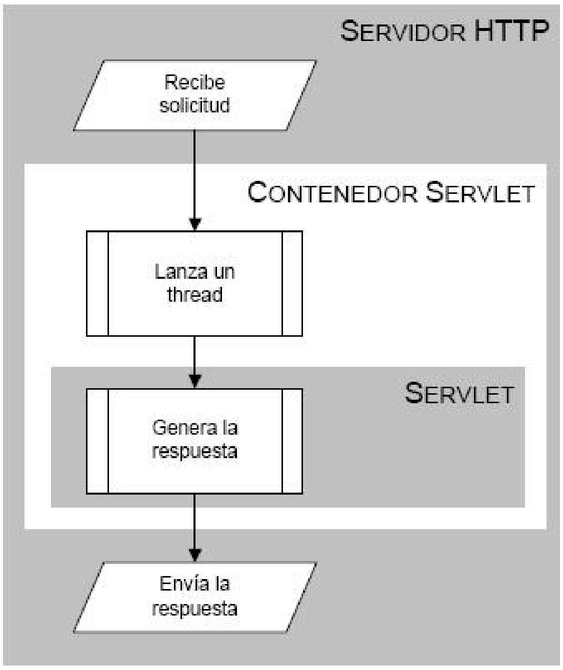


Figura 3: Esquema de Funcionamiento de un servlet

Un servlet cumple un ciclo de vida controlado por el contenedor en el cual se ejecuta. Cuando se hace una petición a un servlet ocurre lo siguiente:

1. Si no existe una instancia del servlet entonces el contenedor web:

    a. Carga la clase del servlet.

    b. Crea la instancia.

    c. Lo inicializa llamando el método INIT.

2. Si existe entonces el contenedor invoca el método service, pasándole los objetos request y response.

Una vez inicializado un servlet este es guardado en memoria. Así, cada vez que llegue una petición, esta va hacia el servlet en memoria, que a su vez genera una respuesta rápidamente.

A continuación se detallan las tareas más comunes que realiza un servlet:

1. Lee cualquier dato enviado por el usuario: comúnmente se introducen por una página web.
2. Obtiene otra información sobre la petición que se encuentra embebida en la propia petición http, como los *cookies*, el nombre del *host* que envió la petición, entre otros.
3. Genera los resultados: se puede requerir el procesamiento y acceso a base de datos, cómputos, entre otros.
4. Genera un documento con los resultados: una página HTML, una imagen, un archivo comprimido, entre otros.
5. Establece los parámetros apropiados para la respuesta.
6. Envía la respuesta al cliente: el documento resultante, con el formato final, se entrega como respuesta al cliente.

En nuestra aplicación Java EE los servlets juegan un papel importante en la lógica de negocio, como parte de la capa de Control del modelo MVC. La funcionalidad general está heredada de las librerías de Struts.

1.4.8. Java Server Pages (JSP)

Una página JSP es un documento que contiene 2 tipos de texto: información estática, representada en cualquier formato basado en texto (como XML, HTML, WML), y los elementos JSP, que construyen el contenido dinámico.

La tecnología JSP es la tecnología que permite a los desarrolladores generar dinámicamente una página web.

Mediante JSP el código Java puede ser embebido en el contenido estático. El código, escrito entre marcas de etiqueta <% y %>, puede acceder fácilmente a los objetos en Java y generar contenido dinámico.

JSP es una mejor solución a la alternativa previa, los servlets. Antes, el contenido dinámico era programado a través de servlets, haciendo el proceso largo y tedioso.

Para hacer una petición (request) a una página JSP, se lo hace como un servlet. Esto se debe a que el ciclo de vida y algunas capacidades de las páginas JSP se determinan por la tecnología Java Servlet. De hecho, un JSP es traducido de forma transparente por el contenedor a una clase Servlet y luego compilado, de tal forma que en ejecución lo que existe es un Servlet. En este caso, el Servlet resultante es un JSP compilado.

Las ventajas incluyen su portabilidad, una relativa facilidad de desarrollo, integración librerías que extienden la funcionalidad y una mejor separación de la Vista y el Controlador (del modelo MVC).

Las páginas JSP integran una parte importante de nuestro proyecto para generar dinámicamente el contenido y una respuesta visual al usuario.

1.4.9. Arquitectura de aplicación MVC (Modelo-Vista-Controlador)

En el paradigma MVC (Modelo-Vista-Controlador) las entradas del usuario, el modelo de aplicación y la retroalimentación visual están explícitamente separados y manejados cada uno por una entidad especializada.

Vista (View).- Maneja la salida textual/gráfica que se presenta al usuario.

Controlador (Controller).- Maneja las entradas del usuario, el procesamiento de datos y la lógica de la aplicación, y delega los resultados al Modelo o la Vista.

Modelo (Model).- Maneja el comportamiento y los datos dentro del dominio de la aplicación. Responde sobre su estado (generalmente a la Vista), y responde a instrucciones (generalmente del Controlador).

Esta separación de tareas beneficia en la reducción de código duplicado, centralizando control y haciendo que la aplicación sea más fácil de modificar.

Existen 2 modelos MVC, el modelo 1 y el modelo 2.

MODELO 1 y MODELO 2

Son dos arquitecturas de diseño de aplicaciones que aparecieron en las primeras especificaciones de JSP. Modelo 1 y Modelo 2 simplemente se refieren a la ausencia o presencia (respectivamente) de un servlet controlador que procesa requerimientos del cliente y selecciona las salidas (páginas de respuesta).

Modelo 1

La aplicación es centrada-en-la-página (page-centric). Cada JSP procesa su propia entrada.

Es fácil de implementar este modelo, orientado a proyectos pequeños. Sin embargo es un modelo no flexible, difícil de mantener, y difícil cuando se desea dividir tareas entre el diseñador y el desarrollador.

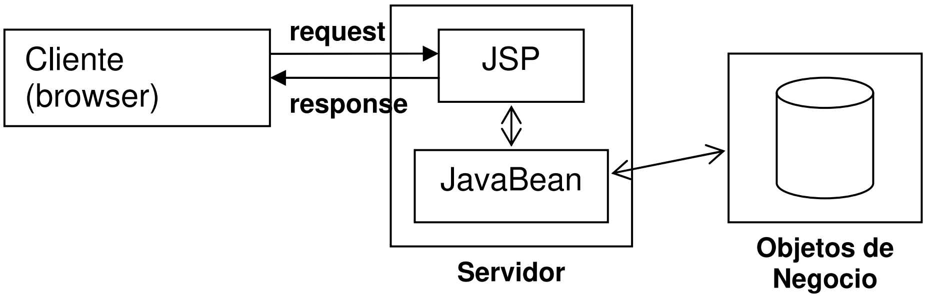


Figura 4: Modelo 1 de arquitectura MVC

Modelo 2

Básicamente separa la generación de contenido de la presentación de contenido.

El servlet controlador despacha requerimientos HTTP[13] a los respectivos JSPs de presentación.

En esta arquitectura las aplicaciones son más fáciles de mantener y más flexibles. El servlet proporciona un control central para la seguridad y el ingreso de usuarios.

Es el modelo estándar para aplicaciones no pequeñas.

---

[13] Requerimiento HTTP generalmente representa la petición de un usuario por el navegador web, aunque no se limita a esa posibilidad.

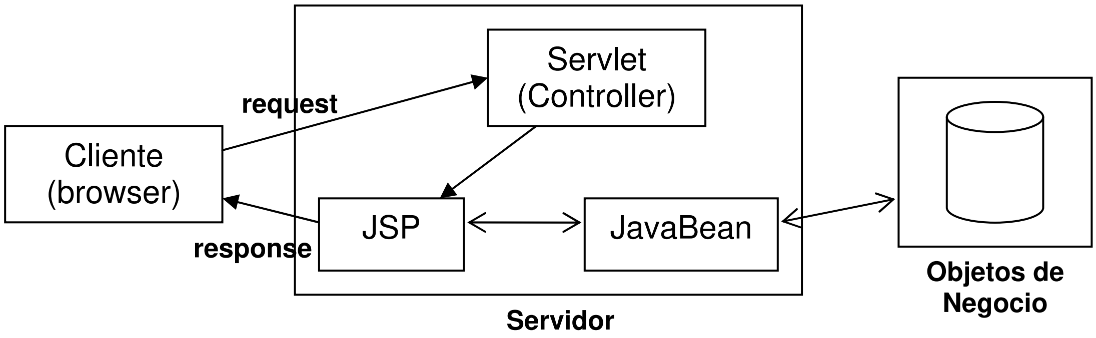


Figura 5: Modelo 2 de arquitectura MVC

CAPÍTULO 2

2. Análisis

En este capítulo haremos un análisis de Eguana Reports, los objetivos, metas y modelo de la aplicación previo al diseño, así como la funcionalidad. También se hará un análisis de las herramientas de desarrollo y su integración.

El análisis, comparado con otros sistemas, puede decirse que se simplifica porque todo gira en torno a la funcionalidad específica del sistema: generar reportes. Desde el punto de vista de un usuario esa es la función del sistema, y las demás funcionalidades están subordinadas a este propósito. Como comparación podemos poner el sistema Eguana E-Procurement, donde una funcionalidad es comprar, otra vender, otra hacer licitaciones, todas con un mismo valor y existen varios actores: comprador, vendedor, administrador, empresa, banco, usuario no registrado, entre otros.

El “generar reportes” es apenas una funcionalidad en otro sistema y en nuestro caso los únicos actores son el usuario interesado en generar reporte, independiente del rol que juegue en otro entorno, y el administrador del sistema.

Esta simplificación a nivel de análisis se compensa con la dificultad técnica en el desarrollo y la integración de herramientas.

2.1. Alcance del proyecto. Una visión desde el punto de vista técnico.

En principio, Eguana Reports se considera como un módulo del sistema E-Procurement[14] llamado Eguana, desarrollado en módulos por los demás integrantes del tópico de graduación. A pesar de esta consideración, Eguana Reports es una aplicación Java EE con funcionalidad específica que se ejecuta de manera independiente y puede integrarse a cualquier aplicación que necesite generar reportes.

Eguana Reports se ha pensado como un servidor de reportes. El sistema Eguana contempla la creación de reportes, y es ahí donde este servidor de reportes se integra para suplir la necesidad.

Un servidor de reportes tiene la misión de proveer a los usuarios una manera fácil de crear y generar reportes, utilizar reportes previamente parametrizados y diseñados, además de permitir hacerlo rápidamente y de manera segura. Un atributo importante también es poder compartir los

[14] E-Procurement busca una convergencia entre proveedores y compradores en una empresa. Es la sistematización de los procesos internos relacionados con la compra de productos de una empresa como el proceso de selección, requerimiento, autorización, pago y compra final.

reportes con otros usuarios afines, por ejemplo, por departamentos de una empresa.

Objetivos

Es una aplicación Java EE que permite generar reportes en varios formatos como PDF, HTML, XLS, CSV y dirigirlos a varios canales. Ej.: cola de impresión, pantalla, disco.

Los objetivos principales son:

- Generación reportes en varios formatos: PDF, HTML, XLS, CSV, XML.

- Uso de variables y parámetros para la generación dinámica de reportes, a través de plantillas (diseños de reportes) previamente validados.

- Soporte de acceso a múltiples fuentes de datos para el uso en la generación de reportes.

- Creación de grupos de reportes y garantía de acceso a cierto grupo de usuarios. Así, más de un sistema puede utilizar la aplicación, o se pueden crear grupos de usuarios-reportes dentro del mismo sistema.

- Independencia de otras aplicaciones Java EE.

- Diseño de aplicación escalable a nivel de programación.

- Disponibilidad permanente de acceso a la generación de reportes.

Se utilizará el paradigma MVC.- Eguana Reports se debe implementar utilizando los frameworks para desarrollo de arquitectura MVC.

2.2. Funcionalidad

La aplicación se divide en pequeños módulos que cumplen una funcionalidad específica. Las funciones deseadas dentro de Eguana Reports son:

- Administración Eguana Reports.- Para agregar fuentes de datos, reportes, usuarios y grupos, y cualquier otra tarea de administración general. Cada una de estas tareas constituye en sí un pequeño módulo dentro de este.

- Creación y validación de plantilla de reporte.- la plantilla es un diseño en xml que se compila para crear un reporte listo para usar. Esto se explicó de manera general en 1.4.5 Jasper Reports – Herramienta de Reportes.

- Acceso a fuentes de datos.- Dependiendo del tipo de fuente de datos, o del modo de acceso (JDBC, JNDI), crea una conexión para que el generador de reportes la use.

- Generación de reportes.- Obtiene una conexión a la fuente de datos y crea un reporte con la información basándose en un diseño de reporte previamente creado y validado.

- Conversión de formato de presentación.- Convertir el formato a HTML, XLS, PDF, CSV.

- Entrega de reporte.- Entrega de reporte por un medio determinado: pantalla o archivo. Esta función se relaciona con la conversión de formato.

- Control de acceso de usuarios y grupos.- Se crean grupos de usuarios que tienen acceso a cierto grupo de reportes. Se debe validar el acceso al momento de querer generar un reporte.

Según esta funcionalidad, tenemos el siguiente esquema de aplicación:

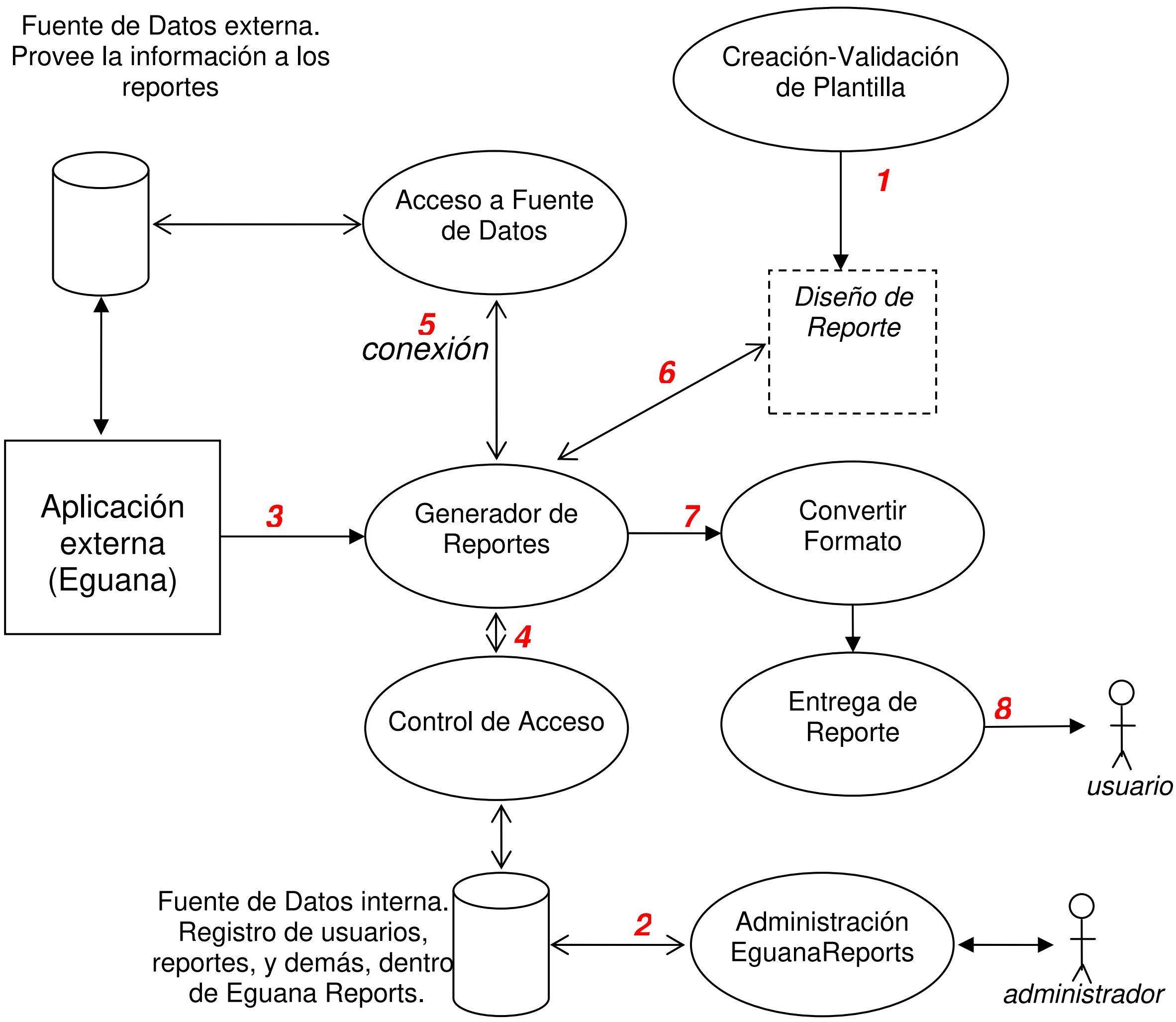


Figura 6: Funcionalidad general de Eguana Reports

El siguiente sería el orden del proceso desde la creación hasta la entrega del reporte al usuario final:

1. Crear una plantilla (diseño de reporte) válida, con el uso de un editor de plantilla (Ej.: iReports, JasperAssistant).

2. La plantilla creada se asocia a un usuario o un grupo que tendrá permiso para su uso en la generación de reporte.

3. El usuario de otra aplicación (por ejemplo Eguana E-Procurement) que desea generar un reporte específico hará una petición al generador de reportes de Eguana Reports, haciendo referencia a un diseño previamente creado en forma de plantilla.

4. Verificando el control de acceso (grupo-usuario-reporte) válido.

5. El generador de reporte obtiene una conexión a la fuente de datos, previamente configurada (JNDI, JDBC), de donde se extrae la información que contendrá el reporte.

6. Basado en el diseño de reporte se genera el reporte.

7. Convierte el formato de presentación del reporte, de ser necesario (a HTML, PDF, XLS, entre otros).

8. Se lo entrega al usuario final.

Este esquema internamente hace uso de varias tecnologías y herramientas Java EE, descritas en el capítulo anterior, para lograr tareas específicas.

2.2.1. Diagrama de clases

El diagrama de clases es un diagrama que muestra la estructura de un sistema. Son utilizados durante el proceso de análisis y diseño para crear el diseño conceptual del sistema, los componentes y la relación entre ellos.

Para el caso de nuestro sistema, todo el flujo de información gira en torno al objeto Reporte, convirtiendo a los demás en objetos secundarios, pero necesarios, en la consecución de la funcionalidad principal del sistema: generar reportes.

El diagrama simplificado de objetos es el siguiente:

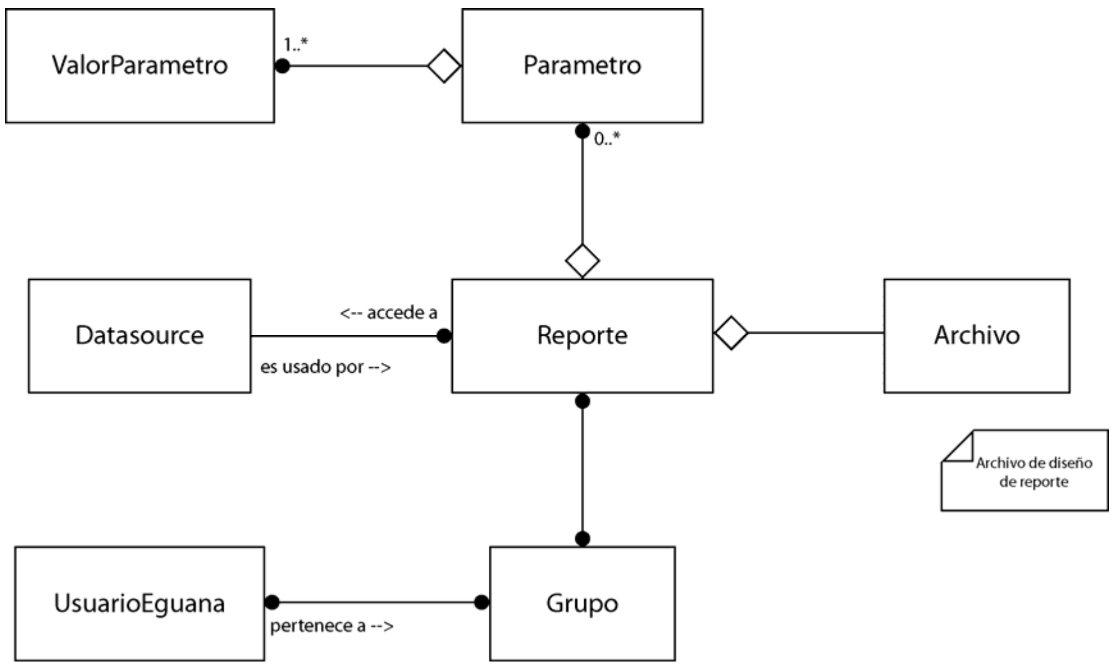


Figura 7: Diagrama simplificado de objetos

Las relaciones entre los objetos se describen así (teniéndo en cuenta las mayúsculas para referirnos a un objeto específico):

- Un `Reporte` puede contener muchos parámetros. A su vez, un `Parámetro` contiene uno o más valores elegibles (`ValorParámetro`).

- `ValorParámetro` es parte de un `Parámetro`. `Parámetro` es parte de un `Reporte`.

- El `Archivo` de diseño es parte del `Reporte`. Si, por ejemplo, se elimina `Reporte`, entonces el `Archivo` (de diseño) relacionado se elimina junto con este.

- Un `Reporte` utiliza una fuente de datos (objeto `Datasource)` para obtener la información para llenar el reporte.

- Una fuente de datos (objeto `Datasource`) puede ser usado por múltiples reportes.

- Un `Reporte` puede ser requerido por muchos grupos (objeto `Grupo`), si se les concede acceso. Un `Grupo` puede tener a su disposición muchos reportes.

- Un `UsuarioEguana` (usuario de Eguana Reports) puede pertenecer a muchos grupos. A su vez, un `Grupo` puede formarse con muchos usuarios.

El diagrama de clases detallado se muestra a continuación. No se detallan las operaciones porque en su mayor parte son operaciones básicas de acceso.

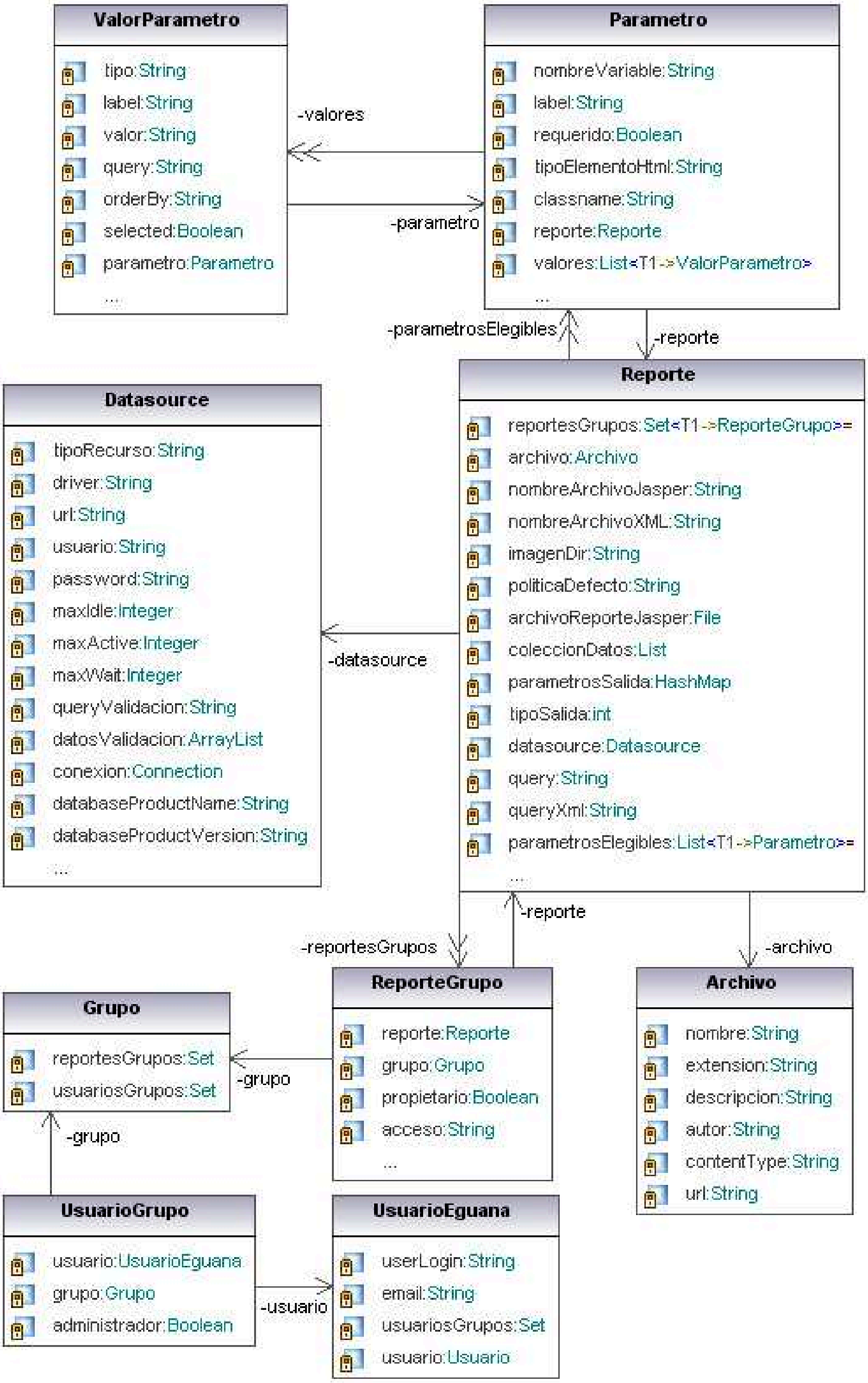


Figura 8: Diagrama de clases

2.2.2. Casos de uso

Un caso de uso es un método utilizado en el análisis, dentro de la ingeniería de software, para describir una funcionalidad o requisito que debe cumplir el sistema. Un sistema posee varios casos de uso. A su vez, un caso de uso posee uno o varios escenarios donde se describen los antecedentes, resultados y la interacción de los objetos y actores dentro de cada escenario.

A continuación el diagrama de casos de uso de Eguana Reports:

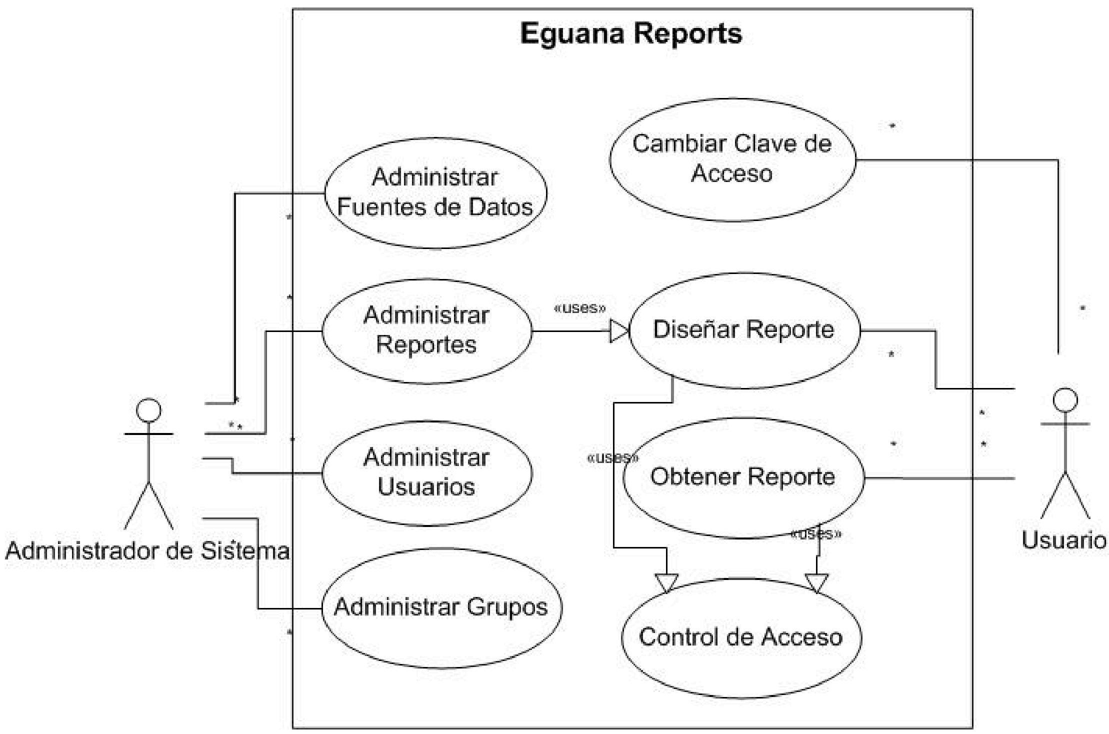


Figura 9: Diagrama de casos de uso

En nuestro caso, hemos definido la administración de fuentes de datos, de usuarios, grupos y reportes como casos de uso. En sistemas con bastantes casos de uso las tareas de administración se dan por sentado y no se presentan de manera explícita. La granularidad (grado de detalle) en este sentido depende del criterio de quien analice el sistema. Para el nuestro, por tratarse de un sistema con pocos casos de uso, hemos aumentado la granularidad en el análisis.

Una vez definidos los casos de uso, detallamos cada uno de ellos:

Caso de uso 1	
Nombre	Administrar Usuarios
Descripción	El administrador del sistema crea, modifica y elimina usuarios.
Actores	Administrador de sistema
Objetivo	Mantener registrado en el sistema a todos los usuarios que pueden obtener reportes.
Notas	Es el usuario quien obtiene un reporte, pero el acceso al reporte es garantizado a partir de grupos de usuarios. Es decir que el usuario debería pertenecer a un grupo que tenga acceso a reportes.

Caso de uso 2	
Nombre	Administrar Grupos
Descripción	El administrador del sistema crea, modifica y elimina grupos. También asocia a los usuarios con los grupos a los que pertenece.
Actores	Administrador de sistema
Objetivo	Mantener registrado en el sistema a todos los grupos de usuarios que pueden obtener reportes.
Notas	Se garantiza acceso a los reportes a través de grupos de usuarios.

Caso de uso 3	
Nombre	Administrar Fuentes de Datos
Descripción	El administrador del sistema crea, modifica y elimina referencias a fuentes de datos.
Actores	Administrador de sistema
Objetivo	Mantener registro de fuentes de datos que estarán disponibles para ser usadas por los reportes.
Notas	Un reporte accede a una fuente de datos a la vez.

Caso de uso 4	
Nombre	Administrar Reportes
Descripción	El administrador del sistema crea, modifica y elimina reportes. También asocia a los reportes con los grupos que tienen acceso.
Actores	Administrador de sistema
Objetivo	Tener disponibles en el sistema reportes ya diseñados y listos para usar por los usuarios.
Notas	Crear, modificar y eliminar, en este punto, se refiere al registro de reporte en Eguana Reports, sus parámetros, permisos y fuente de datos. El diseño del reporte, propiamente dicho, se hace de manera externa o en el caso de uso 5, Diseñar Reporte.

Caso de uso 5	
Nombre	Diseñar Reporte
Descripción	Se diseña el reporte de manera dinámica dentro del sistema con el uso de una herramienta creada para ese propósito.
Actores	Administrador de sistema o usuario con permiso.
Objetivo	Crear dinámica y rápidamente reportes en el sistema y abstraer al usuario de la tediosa tarea de diseñar un reporte.
Notas	El administrador de sistema tiene acceso a todos los reportes. El usuario tiene acceso a los reportes a los cuales previamente se le ha garantizado el permiso.

Caso de uso 6	
Nombre	Obtener Reporte

Descripción	El usuario hace petición para generar un reporte. Si está permitido el sistema genera el reporte y le entrega los resultados.
Actores	Usuario
Objetivo	Generar reportes que satisfagan la necesidad del usuario.
Notas	El reporte debe existir previamente.

Caso de uso 7	
Nombre	Control de Acceso
Descripción	Un usuario que desee ingresar al sistema debe tener credencial de acceso válida, caso contrario se bloquea el acceso.
Actores	Cualquier usuario, directa o indirectamente.
Objetivo	Garantizar la seguridad del sistema y la privacidad de los usuarios.
Notas	

Caso de uso 8	
Nombre	Cambiar Clave de Acceso
Descripción	Un usuario puede cambiar su clave de acceso al sistema.
Actores	Usuario.
Objetivo	Permitir a cualquier usuario cambiar su clave de acceso al sistema, por principios de seguridad.
Notas	Debe verificar la clave anterior antes de cambiarla.

Escenarios

Describiremos los escenarios más representativos:

Escenario 1.1	
Nombre	Creación de usuario exitosa
Descripción	El administrador de sistema desea crear un usuario para que tenga acceso a reportes.
Pre-condiciones	Ninguna.
Post-condiciones	Usuario con credencial de acceso es creado en el sistema.

Escenario 1.2	
Nombre	Modificación de usuario exitosa
Descripción	El administrador de sistema desea cambiar algún atributo de un usuario.
Pre-condiciones	El usuario debe existir.
Post-condiciones	La información modificada del usuario es almacenada en el sistema.

Escenario 1.3	
Nombre	Eliminación de usuario exitosa
Descripción	El administrador de sistema desea eliminar un usuario del sistema.
Pre-condiciones	El usuario debe existir.
Post-condiciones	El usuario es eliminado del sistema y no tendrá acceso futuro.

Escenario 2.1	
Nombre	Creación de grupo exitosa
Descripción	El administrador de sistema desea crear un grupo de usuarios.
Pre-condiciones	Ninguna.
Post-condiciones	El grupo es creado en el sistema.

Escenario 2.2	
Nombre	Modificación de grupo exitosa
Descripción	El administrador de sistema desea modificar la información de un grupo.
Pre-condiciones	El grupo debe existir.
Post-condiciones	La información modificada del grupo es almacenada en el sistema.

Escenario 2.3	
Nombre	Eliminación de grupo
Descripción	El administrador de sistema desea eliminar un grupo del sistema.
Pre-condiciones	El grupo debe existir.
Post-condiciones	El grupo es eliminado del sistema. Los usuarios que pertenecen al grupo no son eliminados del sistema, pero dejan de pertenecer al grupo que antes existía.

Escenario 2.4	
Nombre	Incluir a un usuario en un grupo
Descripción	El administrador de sistema que un usuario pertenezca a un grupo determinado.
Pre-condiciones	El usuario debe existir. El grupo debe existir.
Post-condiciones	El usuario ahora pertenece al grupo.

Escenario 3.1	
Nombre	Creación de fuente de datos exitosa
Descripción	El administrador de sistema desea crear una referencia a fuente de datos de la que se extrae información para llenar un reporte.
Pre-condiciones	Ninguna.
Post-condiciones	En el sistema se almacena la referencia a la fuente de datos.

Escenario 3.2	
Nombre	Modificación de fuente de datos
Descripción	El administrador de sistema modificar la información de la referencia a fuente de datos.
Pre-condiciones	La referencia a fuente de datos debe existir.
Post-condiciones	La información modificada de la fuente de datos es almacenada en el sistema

Escenario 3.3	
Nombre	Eliminación exitosa de fuente de datos
Descripción	El administrador de sistema desea eliminar la referencia a fuente de datos del sistema.
Pre-condiciones	La fuente de datos debe existir. Ningún reporte debe estar utilizando la fuente de datos.
Post-condiciones	La fuente de datos se elimina del sistema.

Escenario 3.4	
Nombre	Probar conexión a fuente de datos
Descripción	El administrador de sistema desea saber si tiene acceso a la fuente de datos.
Pre-condiciones	La fuente de datos debe existir. La información de la fuente de datos es válida.
Post-condiciones	Muestra un mensaje y una prueba de que la conexión a la fuente de datos se pudo realizar.

Escenario 4.1	
Nombre	Creación de reporte exitosa
Descripción	El administrador de sistema o usuario desea crear un reporte.
Pre-condiciones	Debe existir la fuente de datos. Debe existir un archivo de diseño de reporte válido.
Post-condiciones	El reporte es creado y almacenado en el sistema. Se almacena el archivo de diseño.

Escenario 4.1.a	
Nombre	Usuario crea reporte
Descripción	Extensión del escenario 4.1. El usuario desea crear un reporte.
Pre-condiciones	Pre-condiciones 4.1. El usuario debe tener credencial de acceso al sistema. El usuario debe pertenecer a algún grupo.
Post-condiciones	Post-condiciones 4.1 El reporte es asociado al grupo al que pertenece el usuario.

Escenario 4.2	
Nombre	Modificación de reporte exitosa
Descripción	El administrador de sistema o usuario desea modificar la información del reporte.
Pre-condiciones	Debe existir el reporte. Debe existir la fuente de datos. Debe existir un archivo de diseño de reporte válido.
Post-condiciones	La información modificada del reporte es almacenada en el sistema.

Escenario 4.2.a	
Nombre	Modificación de reporte exitosa
Descripción	Extensión del escenario 4.2: El usuario desea modificar la información del reporte.
Pre-condiciones	Debe existir el reporte. Debe existir la fuente de datos. Debe existir un archivo de diseño de reporte válido. El usuario debe tener credencial de acceso al sistema. El usuario debe pertenecer al grupo propietario del reporte.
Post-condiciones	La información modificada del reporte es almacenada en el sistema.

Escenario 4.3	
Nombre	Eliminación de reporte
Descripción	El administrador de sistema o usuario desea eliminar un reporte del sistema.
Pre-condiciones	Debe existir el reporte. Debe existir la fuente de datos. Debe existir un archivo de diseño de reporte válido.
Post-condiciones	El reporte es eliminado del sistema. El archivo de diseño es eliminado del sistema. La asociación del reporte con otros grupos deja de existir.

Escenario 4.3.a	
Nombre	Eliminación de reporte
Descripción	Extensión del escenario 4.3. El administrador de sistema o usuario desea eliminar un reporte del sistema.
Pre-condiciones	Debe existir el reporte. Debe existir la fuente de datos. Debe existir un archivo de diseño de reporte válido. El usuario debe tener credencial de acceso al sistema. El usuario debe pertenecer al grupo propietario del reporte.
Post-condiciones	El reporte es eliminado del sistema. El archivo de diseño es eliminado del sistema. La asociación del reporte con otros grupos deja de existir.

Escenario 4.4	
Nombre	Conceder a un grupo el acceso a un reporte
Descripción	El administrador de sistema desea hacer que un grupo tenga acceso a un reporte.
Pre-condiciones	El reporte debe existir. El grupo debe existir.
Post-condiciones	El grupo tiene ahora acceso al reporte.

Escenario 5.1	
Nombre	Diseño de reporte exitoso.
Descripción	El administrador de sistema o usuario desea diseñar un reporte. Es una función que forma parte de la creación/modificación de reporte. En el proceso, el diseño debe cumplir con las reglas (requisitos) para un diseño válido.
Pre-condiciones	Debe existir la fuente de datos. Debe existir el reporte.
Post-condiciones	Se almacena el archivo de diseño. Se actualiza el reporte para incluir el diseño.

Escenario 6.1	
Nombre	El reporte es obtenido exitosamente
Descripción	El usuario desea obtener un reporte. Hace una petición y el sistema genera el reporte a partir del diseño previamente definido y con la información de la fuente de datos asociada al reporte.
Pre-condiciones	El reporte debe existir y ser válido. La fuente de datos debe estar accesible. El usuario debe tener credencial de acceso al sistema. El usuario debe pertenecer a un grupo con permiso para acceder al reporte.
Post-condiciones	El usuario obtiene el reporte final en el formato deseado.

Escenario 6.2	
Nombre	El reporte no se pudo generar.
Descripción	El usuario desea obtener un reporte. Hace una petición y el sistema hace el intento de generar el reporte a partir del diseño previamente definido y con la información de la fuente de datos asociada al reporte. Falla porque no pudo conectarse a la fuente de datos.
Pre-condiciones	El reporte debe existir y ser válido. La referencia a la fuente de datos existe, pero no se puede establecer conexión. El usuario debe tener credencial de acceso al sistema. El usuario debe pertenecer a un grupo con permiso para acceder al reporte.
Post-condiciones	El reporte no se genera. El usuario obtiene un mensaje indicando que no se pudo establecer conexión con la fuente de datos.

Escenario 7.1	
Nombre	El usuario accede con éxito al sistema.
Descripción	Una persona intenta ingresar como usuario al sistema. El sistema hace petición de una credencial de acceso válida.
Pre-condiciones	El usuario existe. La credencial de acceso del usuario es válida.

Post-condiciones	El usuario tiene acceso al sistema y se le presenta la interfaz visual de usuario.

Escenario 7.2	
Nombre	Se niega acceso al usuario.
Descripción	Una persona intenta ingresar como usuario al sistema. El sistema hace petición de una credencial de acceso válida. La persona ingresa los datos de autenticación, pero el usuario no existe.
Pre-condiciones	El usuario no existe.
Post-condiciones	El usuario recibe un mensaje indicando que se ha negado el acceso.

Escenario 7.3	
Nombre	Se niega acceso al usuario.
Descripción	Una persona intenta ingresar como usuario al sistema. El sistema hace petición de una credencial de acceso válida. La persona ingresa los datos de autenticación, pero la contraseña es inválida.
Pre-condiciones	El usuario existe.
Post-condiciones	El usuario recibe un mensaje indicando que se ha negado el acceso.

Escenario 8.1	
Nombre	El usuario cambia la clave de acceso
Descripción	Un usuario desea cambiar su clave de acceso al sistema. Ingresa su clave anterior y la nueva.
Pre-condiciones	Escenario 7.1.
Post-condiciones	El usuario recibe un mensaje indicando que su nueva clave ha sido registrada.

2.2.3. Diagrama de interacción de objetos

Los diagramas de interacción de objetos muestran cómo interactúan los objetos a través del tiempo para un escenario (contexto) específico. Es una metodología para modelar las partes dinámicas de un sistema.

En este diagrama se muestran instancias de objetos que intervienen en el desarrollo “paso a paso” de un escenario correspondiente a un caso de uso. De aquí se conoce los mensajes que envía qué objeto a otro, en qué momento y cuál es la respuesta esperada, y, en el contexto general, cómo se llega a obtener los resultados descritos en el escenario (post-condiciones).

Según el modelo MVC existe una capa de control que interactúa con la capa de vista. La capa de control se delega a varios objetos según el caso de uso, y en cada escenario interviene uno sólo de acuerdo al contexto. Por ejemplo, para el escenario 1.1 Creación de usuario exitosa interviene el objeto que llamaremos ControlUsuarios, y es el punto de partida para la lógica de negocio en este escenario. En implementación estos objetos heredan funcionalidad que proporciona Struts.

Así también, la capa de modelo es representada por un objeto que llamaremos EguanaDAO (en relación a Data Access Object para Eguana Reports). En la fase de implementación esta funcionalidad es proporcionada por Hibernate.

A continuación presentamos los diagramas de interacción de los escenarios más representativos del sistema.

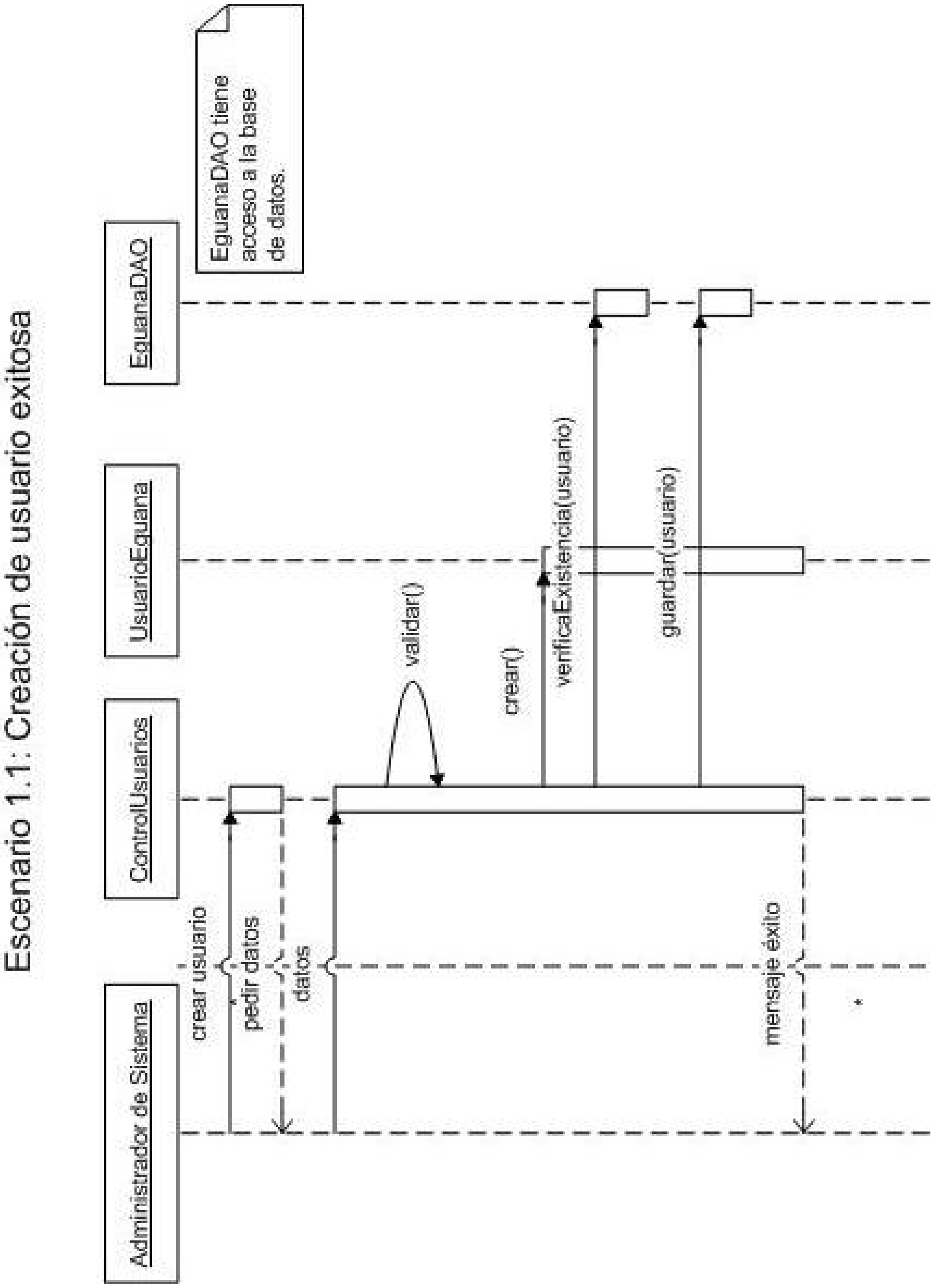


Figura 10: Escenario 1.1: Creación de usuario exitosa

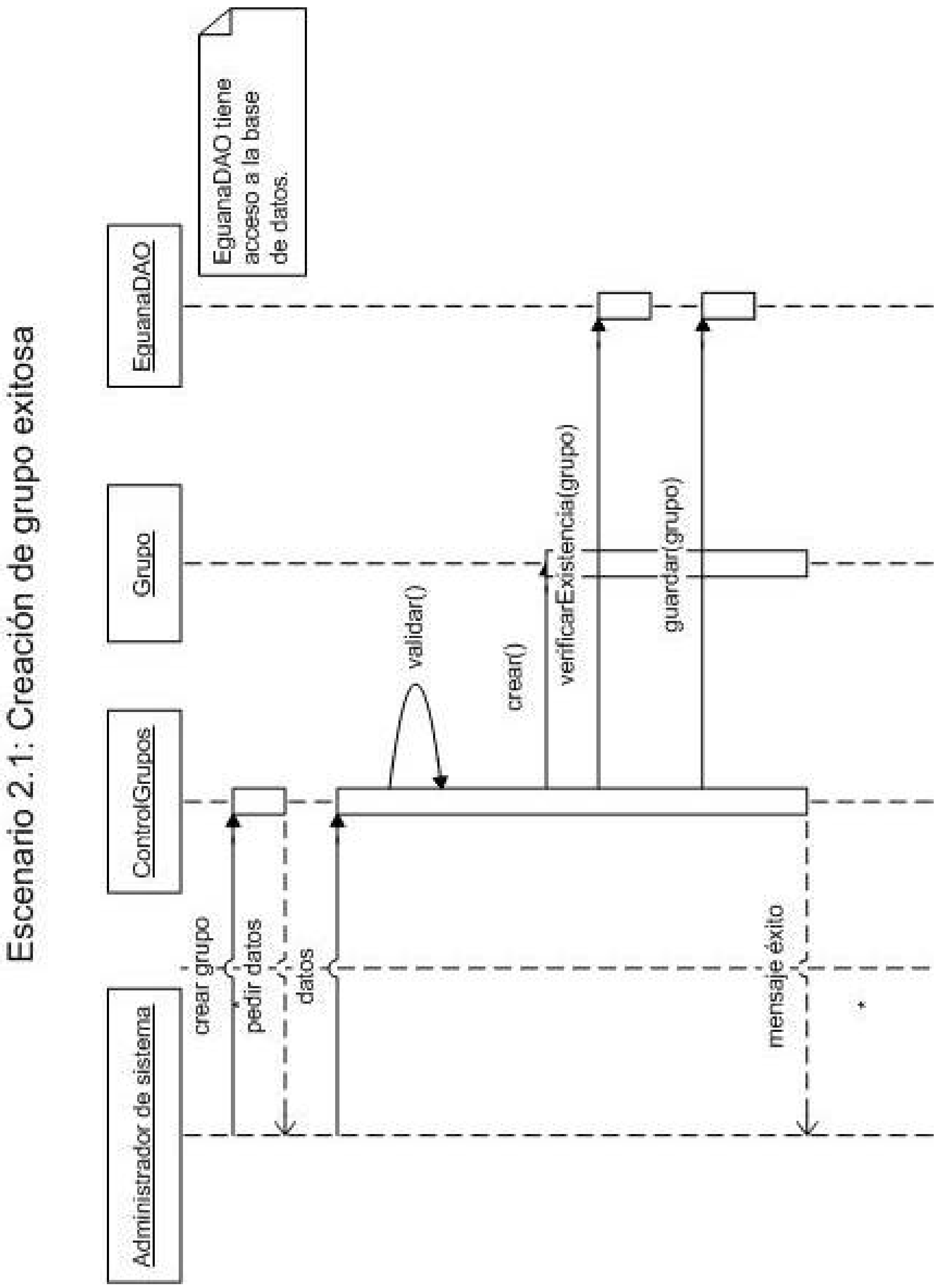


Figura 11: Escenario 2.1: Creación de grupo exitosa

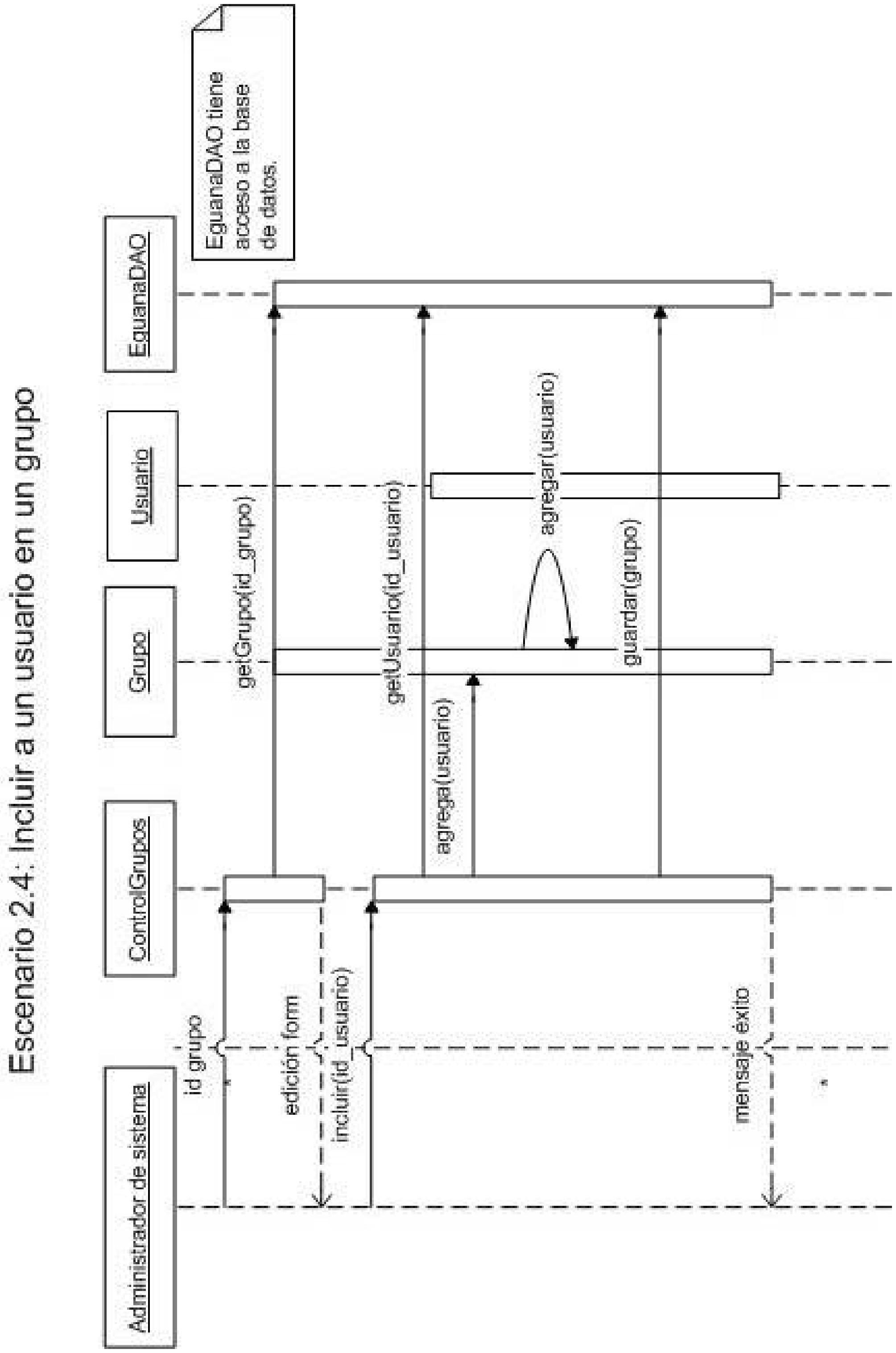


Figura 12: Escenario 2.4: Incluir a un usuario en un grupo

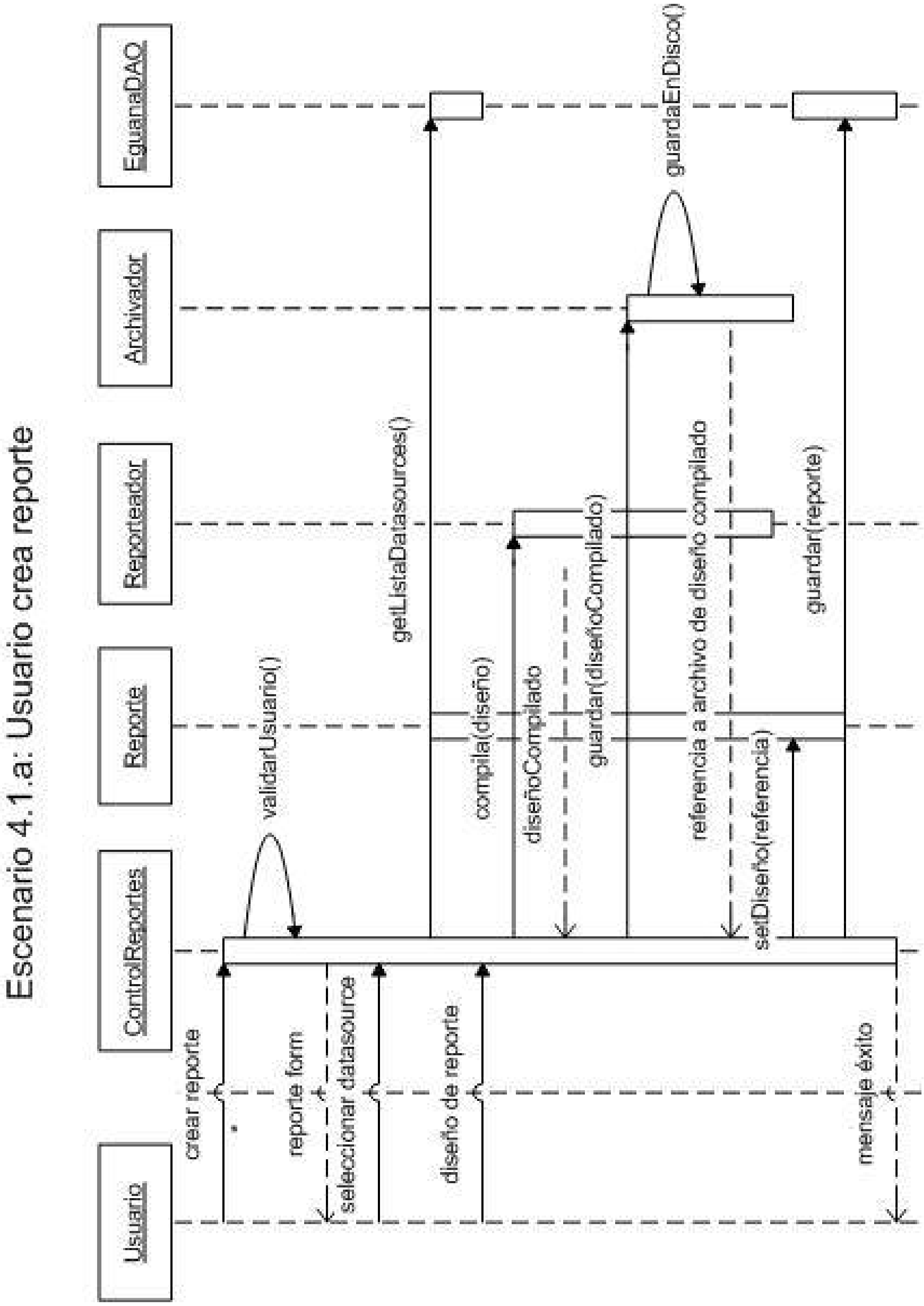


Figura 13: Escenario 4.1.a: Usuario crea reporte

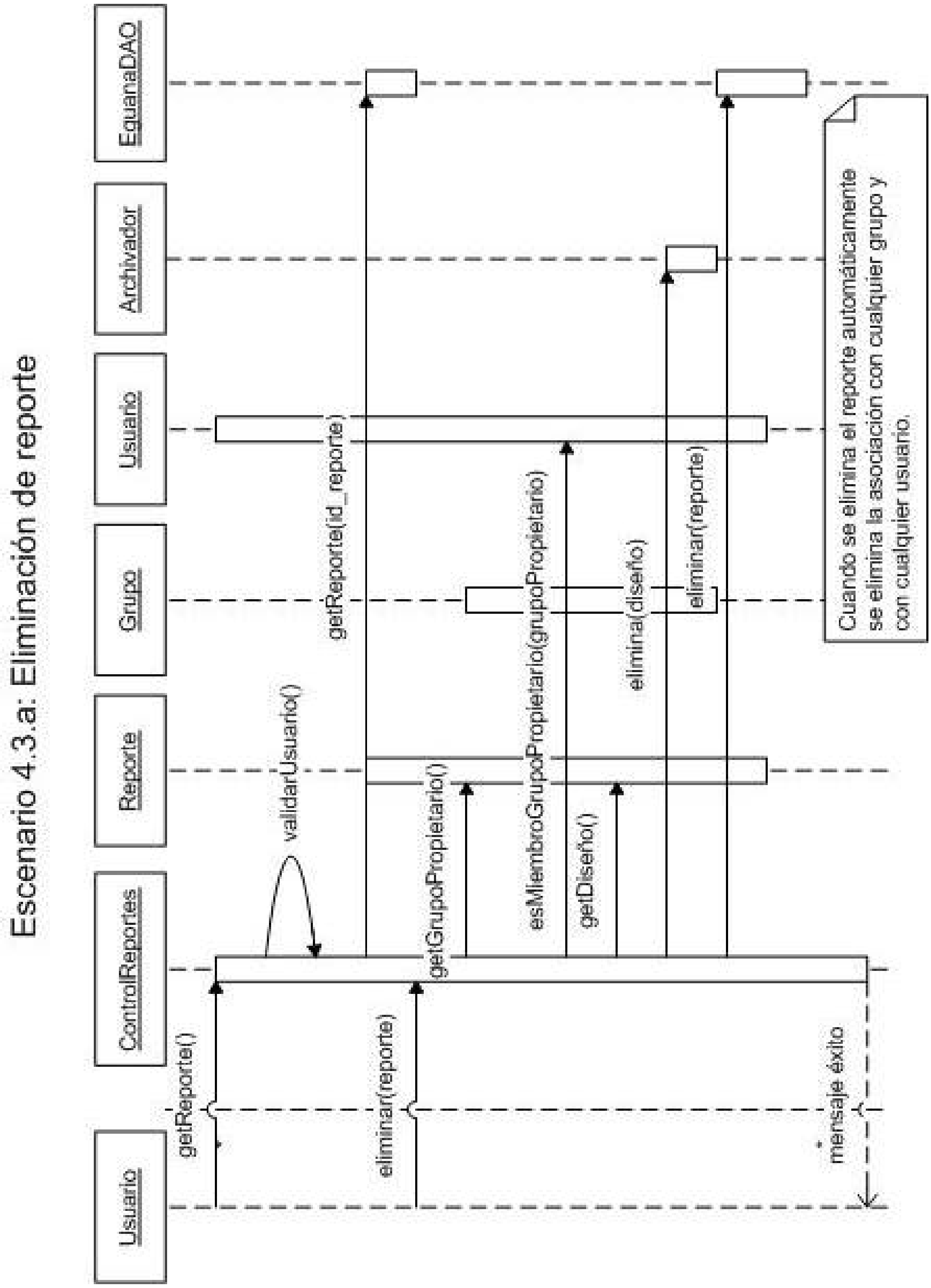


Figura 14: Escenario 4.3.a: Eliminación de reporte

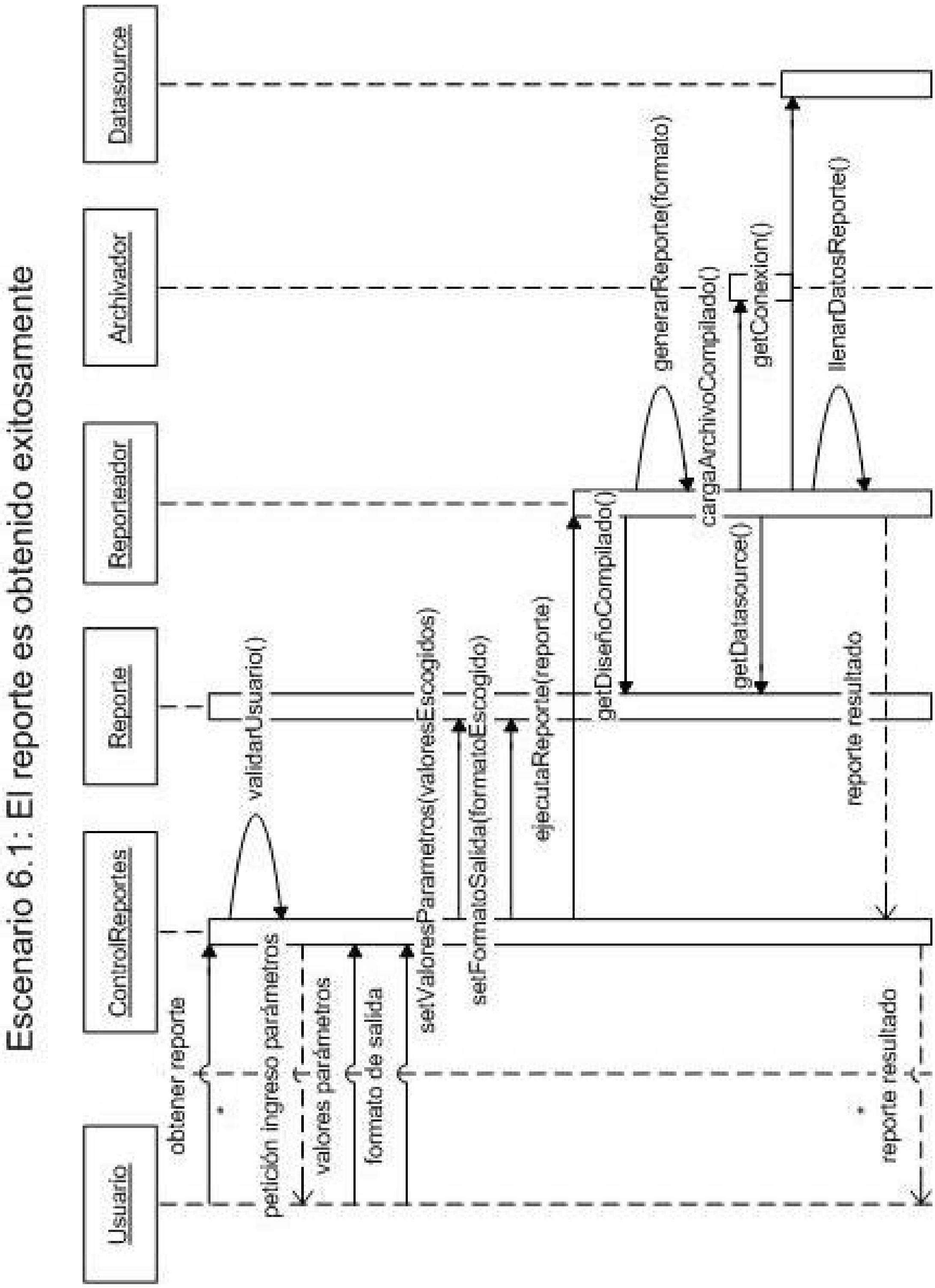


Figura 15: Escenario 6.1: El reporte es obtenido exitosamente

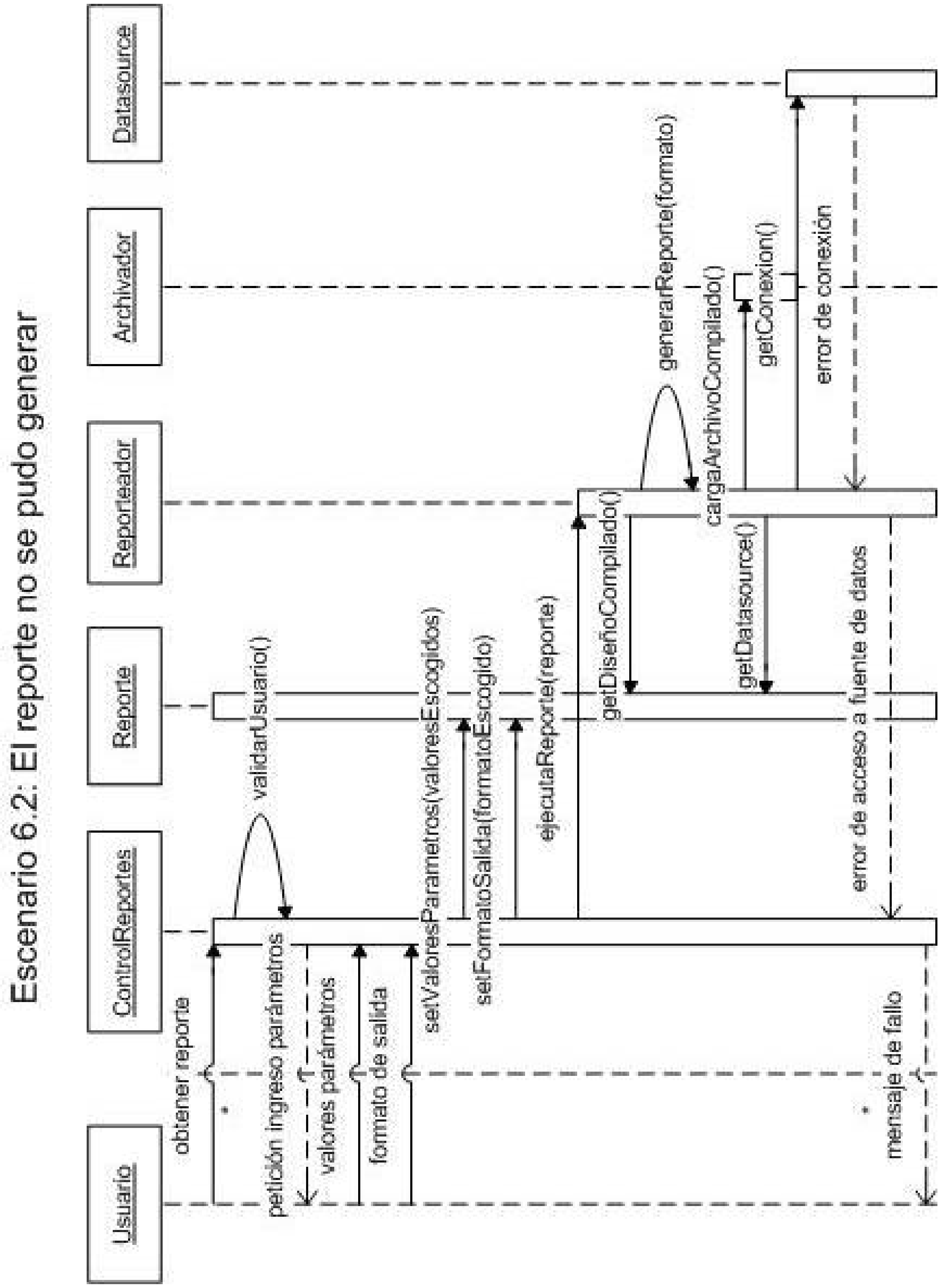


Figura 16: Escenario 6.2: El reporte no se pudo generar

2.2.4. Modelo Entidad-Relación

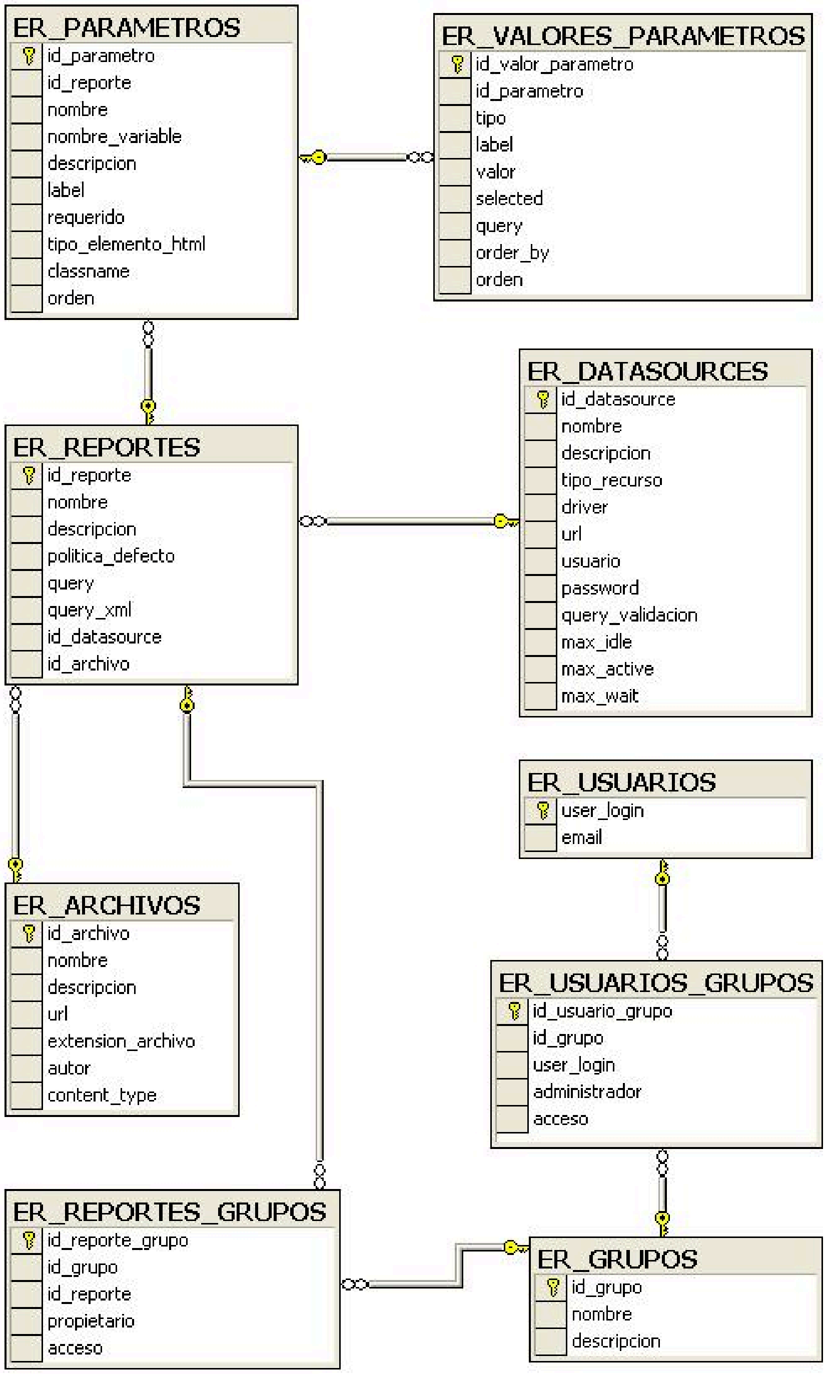


Figura 17: Modelo Entidad-Relación

2.3. Análisis de convergencia de versiones de herramientas de trabajo

Uno de los retos que plantea el análisis, que se extiende hasta las fases de diseño e implementación, es encontrar las herramientas más adecuadas para llevar a cabo el desarrollo de un sistema.

Según nuestra percepción, este reto es más evidente en el entorno de desarrollo de aplicaciones en código abierto. La disponibilidad de herramientas es una de las ventajas, pero a veces encierra la dificultad de elección ante tantas opciones. Y dentro de estas opciones cada herramienta tiene sus versiones.

Nuestras principales guías para esta tarea son las siguientes:

- La opinión y experiencia de una persona con conocimientos avanzados en el tema. En nuestro caso los más directos son nuestros instructores de tópico.

- La búsqueda de sitios web de fabricantes reconocidos por la calidad de sus productos para el desarrollo de aplicaciones, tales como Sun y Apache.

- La búsqueda de sitios web que se dediquen a proveer herramientas, aplicaciones, código abierto. El más conocido de estos sitios es SourceForge.

- La búsqueda de documentos de análisis de productos disponibles en Internet.

- Analizar el ranking de productos en el mercado.

- Leer y analizar, si existe, la información que provee el propio fabricante de la herramienta. Este suele decir con qué herramientas se integra mejor y con qué versiones. Incluso suele proveer un paquete donde se incluyen todas estas herramientas listas para utilizar.

- Una de las más importantes es analizar la disponibilidad de documentación (para instalación y configuración) y mecanismos de soporte y ayuda, ya sea a través de foros, correo, publicaciones o documentos.

- Por último, tal vez la mejor guía es analizar el porcentaje de descarga en relación a otras herramientas y la opinión de los usuarios.

2.3.1. Eclipse como herramienta de desarrollo

En términos simples, Eclipse es la herramienta de desarrollo de aplicaciones más extendida en la comunidad de código abierto.

Nació a mediados de los años 90, cuando Microsoft Visual Studio se convertía en una herramienta de desarrollo de propósito general. IBM creó Eclipse con el propósito de establecer una plataforma común para los desarrolladores y evitar el duplicado de elementos comunes en la infraestructura de la aplicación. Desde el inicio IBM se centró en la visualización del entorno de desarrollo como un ambiente compuesto de múltiples herramientas en combinación.

Al inicio Eclipse pretendía dos cosas: proveer soporte de ejecución para Microsoft y la otra centrarse en una industria de desarrollo más abierta basada en Java. Es esta última la que le aseguraría el éxito a largo plazo.

Eclipse ha sido por mucho tiempo sinónimo de “desarrollo en Java”, o Java IDE[15], aunque actualmente ha crecido para convertirse en una comunidad (la comunidad Eclipse) que provee una plataforma de desarrollo expandible, que pueda ser usada a lo largo del ciclo de un proyecto. Provee herramientas

[15] IDE (Integrated Development Environment) Ambiente Integrado de Desarrollo

para soporte de desarrollo en C/C++, PHP, web, además de UML, XML, bases de datos, entre otros.

Sobre la plataforma básica de Eclipse se instalan las herramientas de desarrollo en forma de componentes (plugins). Esto convierte a Eclipse en una plataforma muy flexible, configurable y escalable. La comunidad Eclipse clasifica los plugins en más de 20 categorías y existen más de 1000 plugins disponibles.

Eguana Reports utiliza la versión de Eclipse 3.1.2.

Además utilizaremos dos plugins:

- MyEclipse 4.1.0: Es un plugin de licencia comercial que cuesta cerca de USD 30.00, una pequeña inversión con respecto a su utilidad. Provee herramientas para soporte de Struts, configuración de Hibernate, además de tareas generales en el momento de escribir código.

- FileSync 1.2.2: Es un pequeño plugin que nos permitirá mantener la sincronía de los archivos entre el espacio de trabajo y el contenedor web. No es necesario, pero a nosotros nos resulta útil para hacer cambios en caliente[16] y acelerar un poco la codificación.

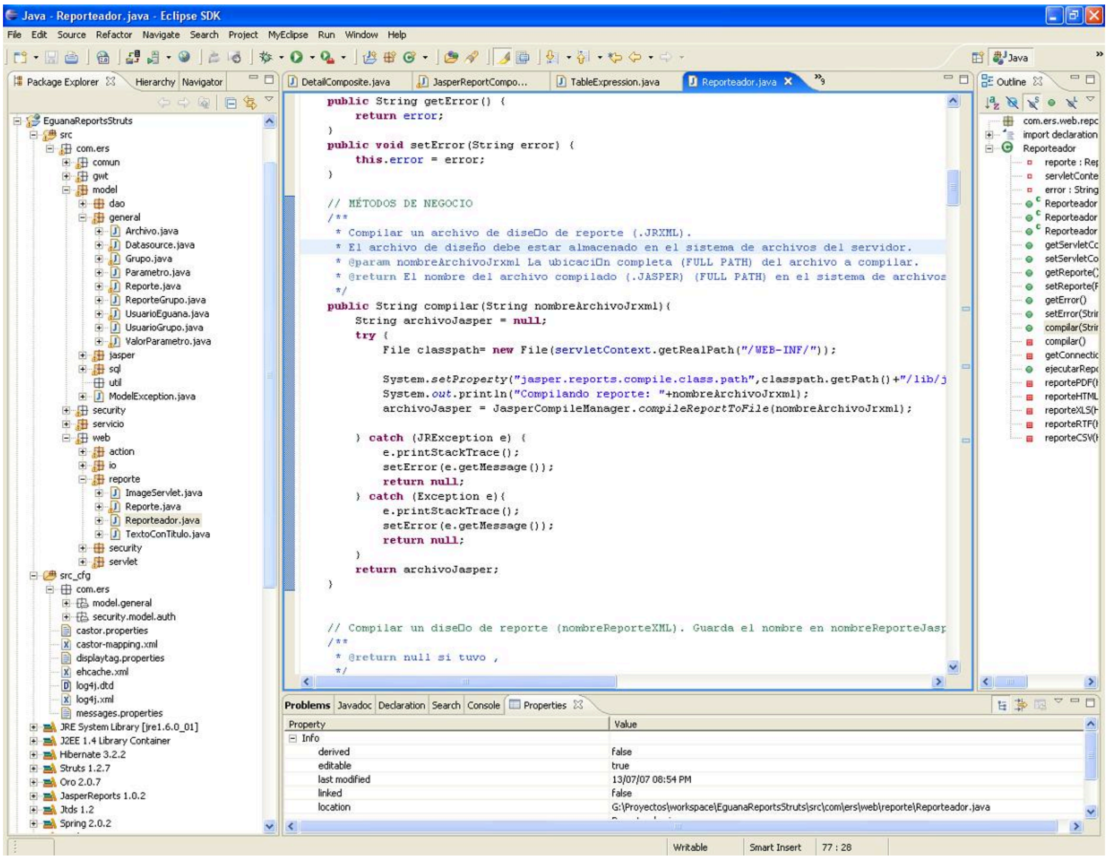


Figura 18: Pantalla de Eclipse

2.3.2. Integración entre servidor, aplicación y base de datos

Recordemos el modelo de aplicación multicapas que define Java EE. Simplificando nuestra aplicación para entender en breves rasgos qué parte corresponde a qué capa, tenemos lo siguiente:

---

[16] Que los cambios se reflejen inmediatamente en la aplicación que se está ejecutando.

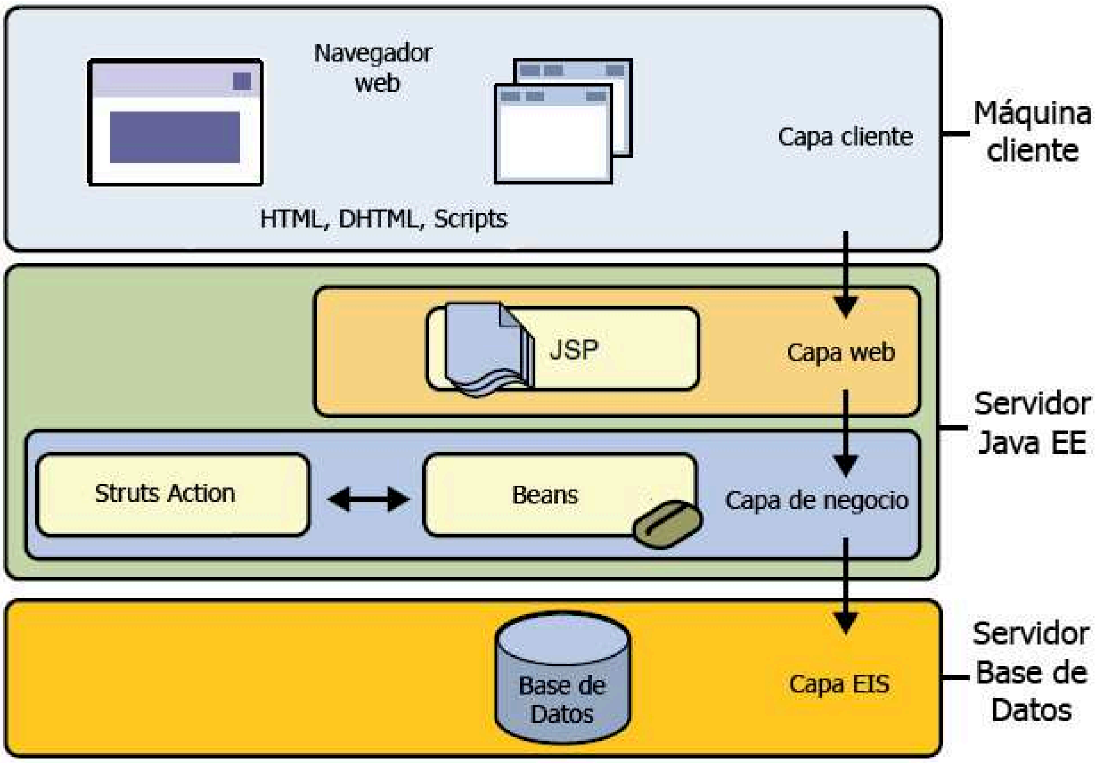


Figura 19: Modelo de aplicación Java EE aplicado a Eguana Reports

El usuario interactúa con nuestra aplicación haciendo uso de un navegador web.

En el servidor se generan los JSP para entregar la interfaz visual al usuario. Al mismo tiempo, los requerimientos del usuario que corresponden al negocio son procesados por los objetos de control de Struts, que interactúa con los beans [17] para luego guardarlos en la base de datos haciendo uso de Hibernate.

2.3.3. Integración de herramientas para implementar arquitectura MVC

Nosotros utilizaremos Eclipse 3.1 como base para el desarrollo y programación, y le instalamos MyEclipse 4.1.0 como principal soporte para implementar la arquitectura MVC.

Cuando se trata plugins pequeños necesitamos copiar las librerías (del plugin) dentro de la carpeta “plugins” de Eclipse, con Eclipse previamente cerrado. En este caso no es así. MyEclipse provee un instalador ejecutable que nos guía durante la instalación:

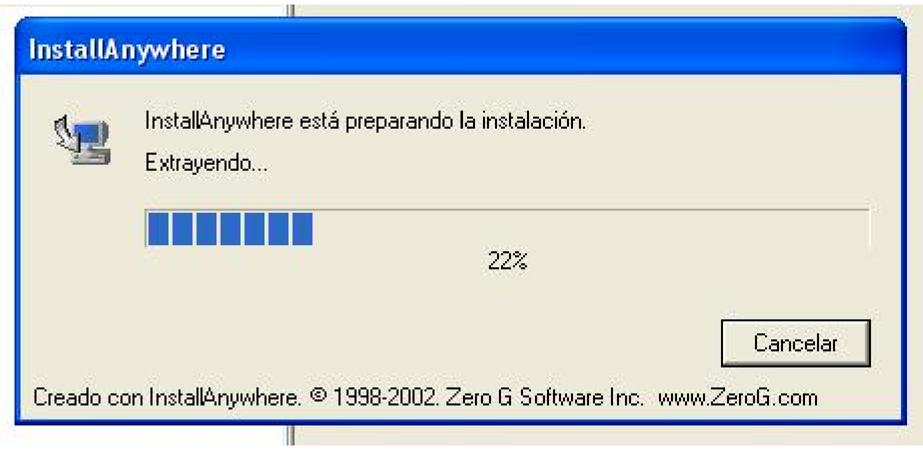


Figura 20: Ejecutando instalador de MyEclipse plugin

[17] Objetos representativos del negocio que encapsulan y modelan los datos.

Lo único que debemos hacer es indicar la carpeta de instalación de Eclipse:

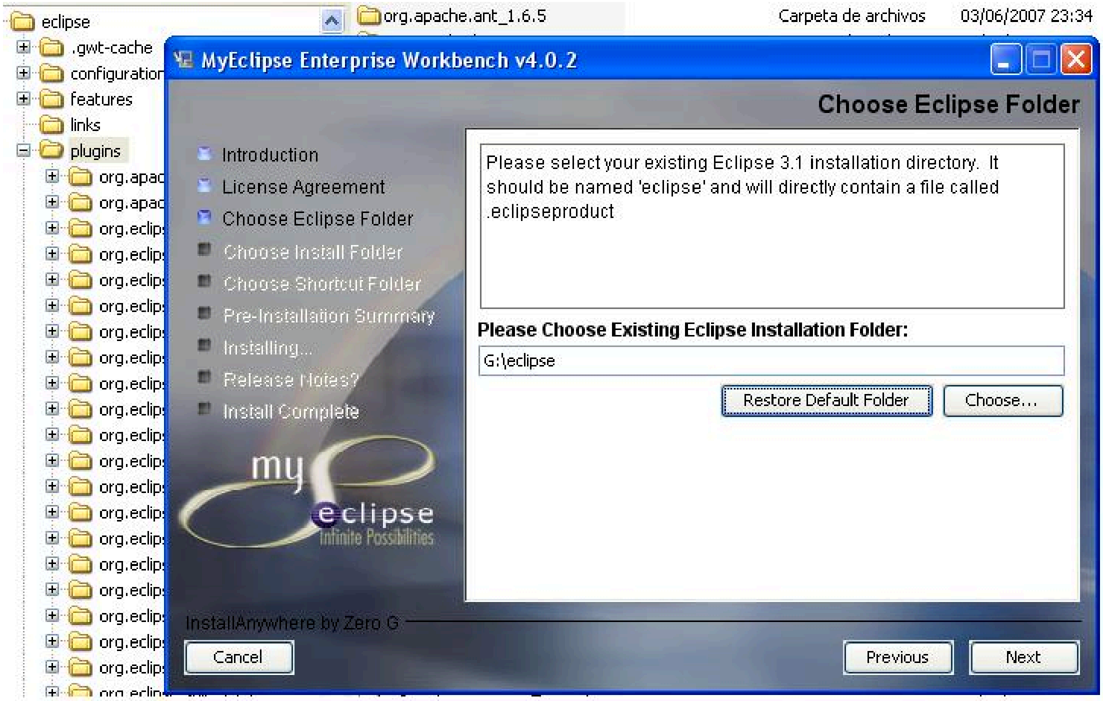


Figura 21: Instalando MyEclipse plugin

Eso es todo. Ya tenemos la herramienta para implementar MVC a la mano.

En el próximo capítulo de analiza más en detalle el modelo de arquitectura MVC.

2.3.4. Interacción entre Jasper Reports y Castor

El DTD de JasperReports para la construcción del diseño de reporte, que al final es un documento XML con extensión .JRXML, es convertido inicialmente a XSD[18]. A partir de este XSD creamos (haciendo unmarshal) una estructura de objetos. Esta operación la hacemos una sola vez con el objetivo de utilizar más adelante esos objetos en la herramienta de diseño dinámico de reporte.

Haciendo uso de Castor mismo, y con la estructura de objetos en la aplicación, podemos crear instancias de diseño de reporte y leerlas. Es un camino rápido para ir desde la interfaz gráfica hasta el documento XML.

El diseño de reporte es utilizado por JasperReports para generar los reportes.

[18] XML Schema Definition, otro lenguaje para definir los elementos legales en la construcción de un documento XML.

CAPÍTULO 3

3. Diseño – Arquitectura del Servidor de Reportes

Una vez más retomamos el concepto de arquitectura MVC. El término tiene sus inicios por los años 1978 y 1979 en Xerox PARC. Inicialmente contenía cuatro términos Modelo-Vista-Controlador-Editor. El Editor era un componente que hacía referencia a la necesidad de una interfaz entre la Vista y los dispositivos de entrada como teclado y ratón. Los aspectos de entrada y salida (input/output) del Editor fueron más tarde combinados de tal forma que el Editor pasó a contener dos objetos: Vista y Controlador.

La Vista se encarga de la presentación y el Controlador se encarga de leer e interpretar las entradas del usuario[19].

Incluso se manejó otro término, Herramienta (Tool), que es responsable de dar al usuario la posibilidad de realizar una o más tareas para interactuar con el modelo.

---

[19] Definición según Smalltalk-80 MVC.

La primera versión de MVC fue implementada para la librería de clases de Smalltalk-80.

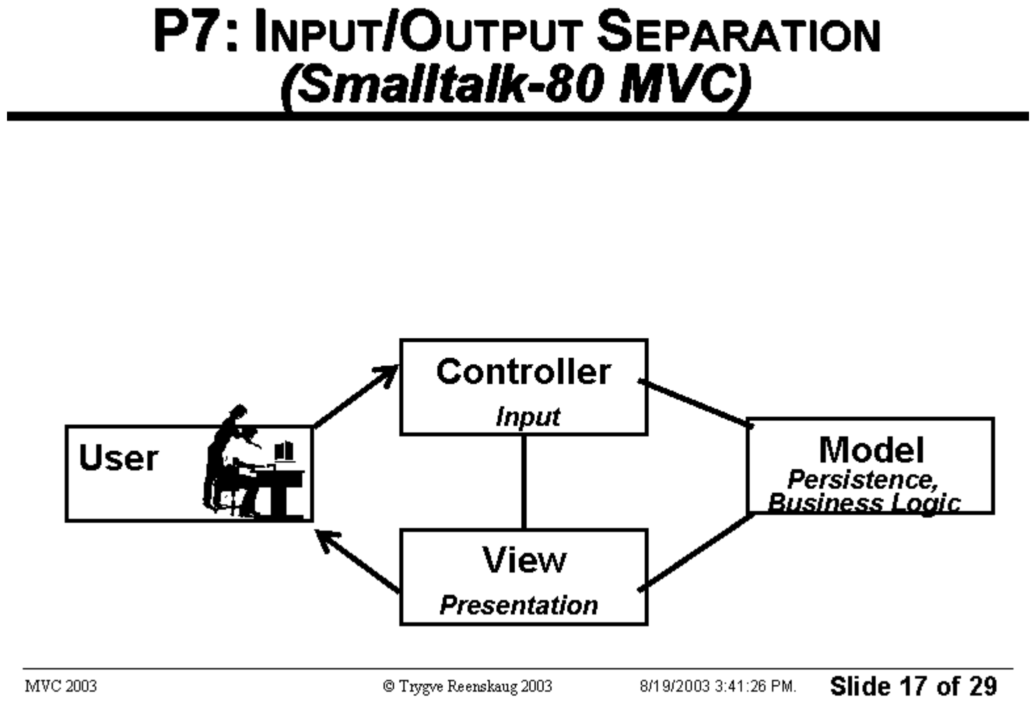


Figura 22: Smalltalk-80 MVC
Fuente: The Model-View-Controller (MVC), Its Past and Present, pág.10
copyright ©2003 Trygve Reenskaug, Oslo, Norway.

En fin. MVC fue concebido como una solución general al problema de usuarios controlando grandes y complejas estructuras de datos. El propósito esencial es reducir la brecha entre el modelo mental del humano (usuario) y el modelo digital que existe en la computadora.

La solución MVC ideal crea la ilusión de ver y manipular directamente la información (modelo). El modelo ha evolucionado a la fecha.

3.1. Arquitectura de la Aplicación Web

Véase como referencia la Figura 19: Modelo de aplicación Java EE aplicado a Eguana Reports.

Puede resultar un poco confuso el mencionar al modelo de aplicación Java EE y modelo de arquitectura MVC y tratar de relacionarlos. El modelo de aplicación Java EE define un modelo de N-capas para una aplicación distribuida, un concepto desde el punto de vista general de una aplicación de acuerdo a la funcionalidad y al lugar de ejecución de cada capa. Aquí se incluye la base de datos como capa, y la aplicación del cliente como parte de otra capa. La arquitectura MVC define un modelo para separar los datos de una aplicación y mantener una estructura interna ordenada (en tres partes: modelo, vista y controlador).

Podemos decir, tal vez, hasta cierto punto, que el modelo Java EE engloba más que la arquitectura. El modelo MVC no discrimina las capas del modelo Java EE, pero sí pretende separar la interfaz de usuario de la lógica de negocio y el acceso a datos. Extendiendo el modelo de aplicación Java EE, un resumen de cómo se combinan estos conceptos se muestra a continuación:

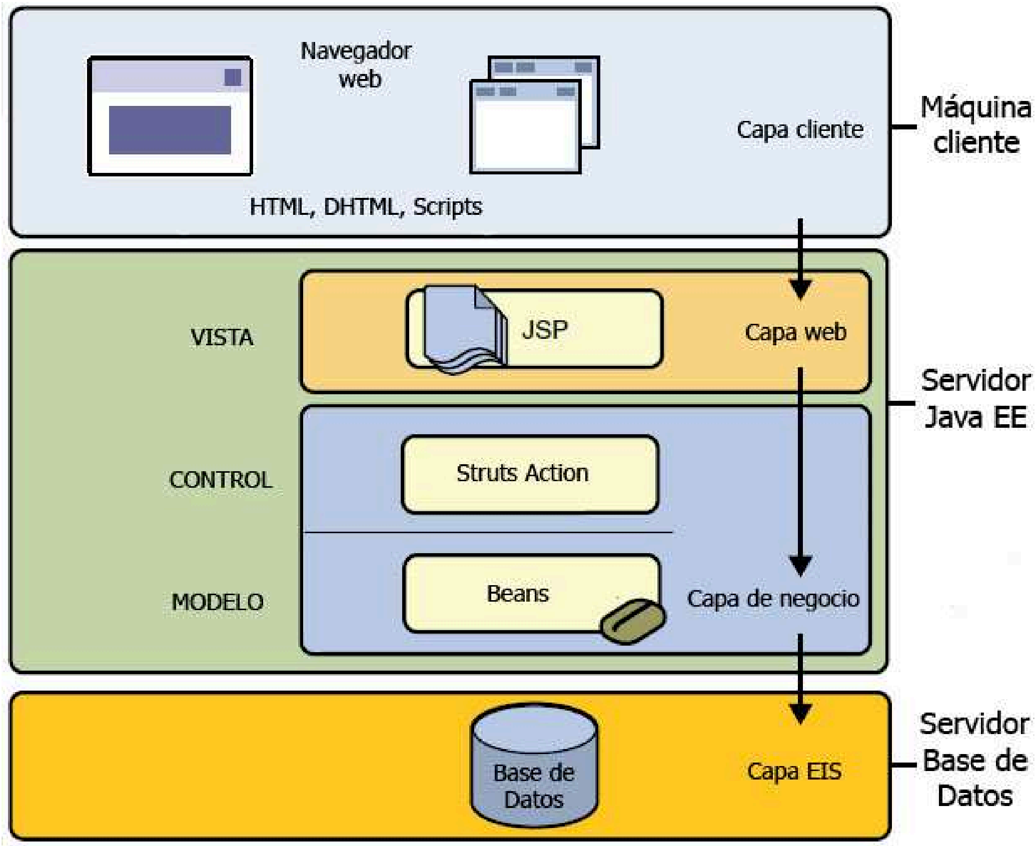


Figura 23: MVC dentro de modelo de aplicación Java EE

3.2. Capa de persistencia y modelo

El modelo es la forma digital de representar la realidad. Por un lado está la representación mental de la realidad, tal como la percibe la persona, y por el otro la representación digital (objetos y partes) del modelo concebido por la persona.

MVC no hace referencia expresa de los mecanismos para almacenar el modelo en el servidor de información (base de datos). Los objetos de acceso a los datos no constituyen el modelo dentro de MVC.

Tal es la razón por la que hacemos una diferencia entre persistencia y modelo. Nuestro modelo corresponde a los objetos `Reporte`, `Datasource`, `Usuario`, `Grupo`, y los demás. La persistencia corresponde a los objetos de acceso a datos (`EguanaBDManager` y `EguanaDAO`) y la tecnología de acceso (Hibernate).

`EguanaBDManager` es nuestro objeto encargado de administrar las tareas con la base de datos. Es quien interactúa directamente con los objetos de control y lógica de negocio; además, es la interfaz entre éstos y `EguanaDAO`.

`EguanaDAO` se encarga de interactuar directamente con la base de datos, hacer consultas y utilizar las librerías que provee Hibernate. No se preocupa de nada más, ni de interactuar con nadie más que `EguanaBDManager`. Es este último quien le entrega los requerimientos filtrados y depurados.

Es la capa de persistencia y modelo en donde se hace la representación de los datos sobre los cuales opera la aplicación. Es la encargada de salvaguardar la integridad de los datos.

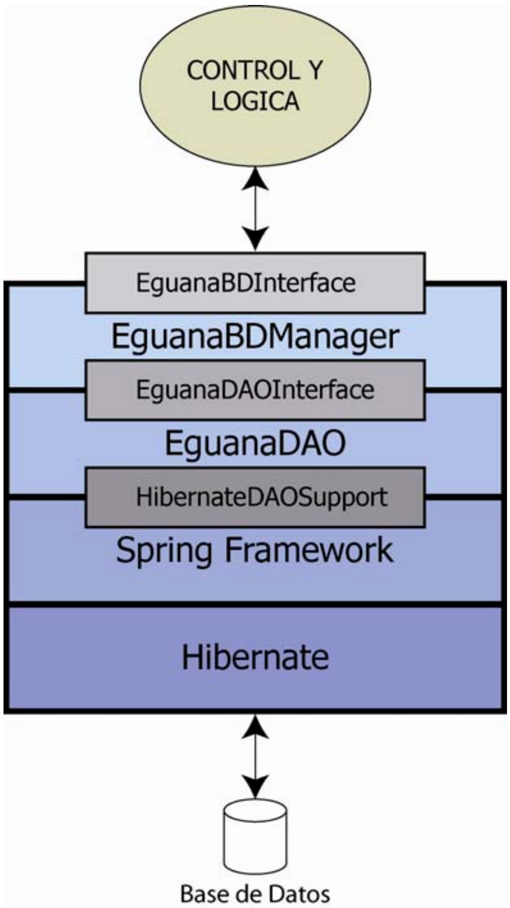


Figura 24: Capa de persistencia

En esta etapa daremos a conocer que las transacciones, creación de sesiones y otras tareas pueden ser abstraídas con el uso de Spring Framework. Spring ofrece también sus propias librerías para soportar MVC, pero esa funcionalidad la tenemos con Struts. Lo que nos interesa es la utilidad para facilitarnos ciertas tareas con Hibernate.

3.3. *Capa de vista*

Vista, en el concepto original de MVC, es la representación visual del modelo. Actúa además como un filtro de presentación, al realzar ciertos atributos del modelo y suprimir otros.

Luego evolucionó para también aceptar entradas de usuario que sean relevantes a la propia Vista.

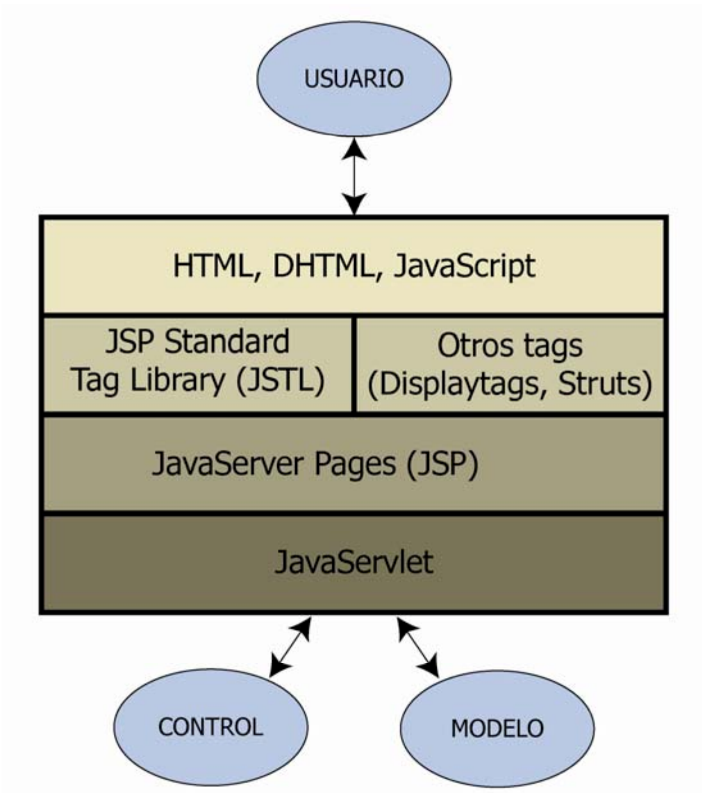


Figura 25: Capa de vista

3.4. *Capa de lógica y control*

Un aspecto importante del modelo MVC original es que el Controlador es responsable de crear y coordinar las Vistas subordinadas. Además es responsable de interpretar las entradas del usuario.

El concepto evolucionó para luego decir que el Controlador acepta e interpreta las entradas del usuario relevantes a Vista/Controlador como un todo, dejando que la Vista se encargue de las entradas relevantes a sí misma.

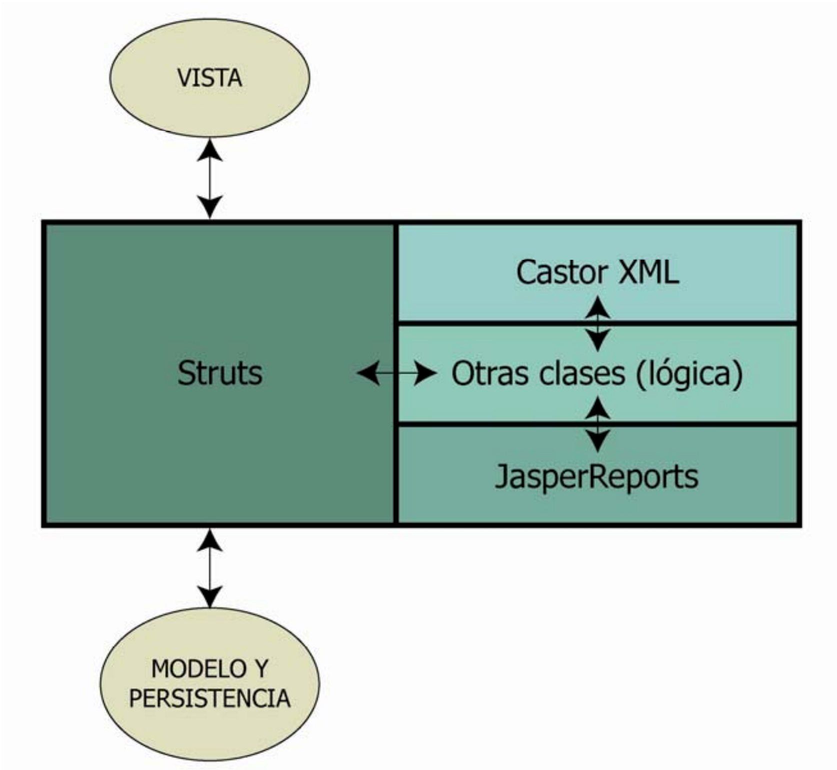


Figura 26: Capa de lógica y control

En esta capa no existen niveles jerárquicos. Struts, que representa al Control, interactúa con la Vista. Interpreta los requerimientos del usuario y delega a los respectivos componentes de lógica de la aplicación.

Entre otras cosas, la parte lógica se encarga de interactuar a su vez con la tecnología Castor XML y con JasperReports para las tareas que conciernen a los reportes.

Esto ya lo explicamos en la sección 2.3.3 Integración de herramientas para implementar arquitectura MVC y en la sección 2.3.4 Interacción entre Jasper Reports y Castor.

3.5. Cómo se realiza la comunicación entre los componentes de la aplicación

Cada componente interactúa con las demás y se integra a un todo, la aplicación Java EE.

Ahora veamos de forma resumida cómo interactúan las diferentes tecnologías y componentes dentro de Eguana Reports:

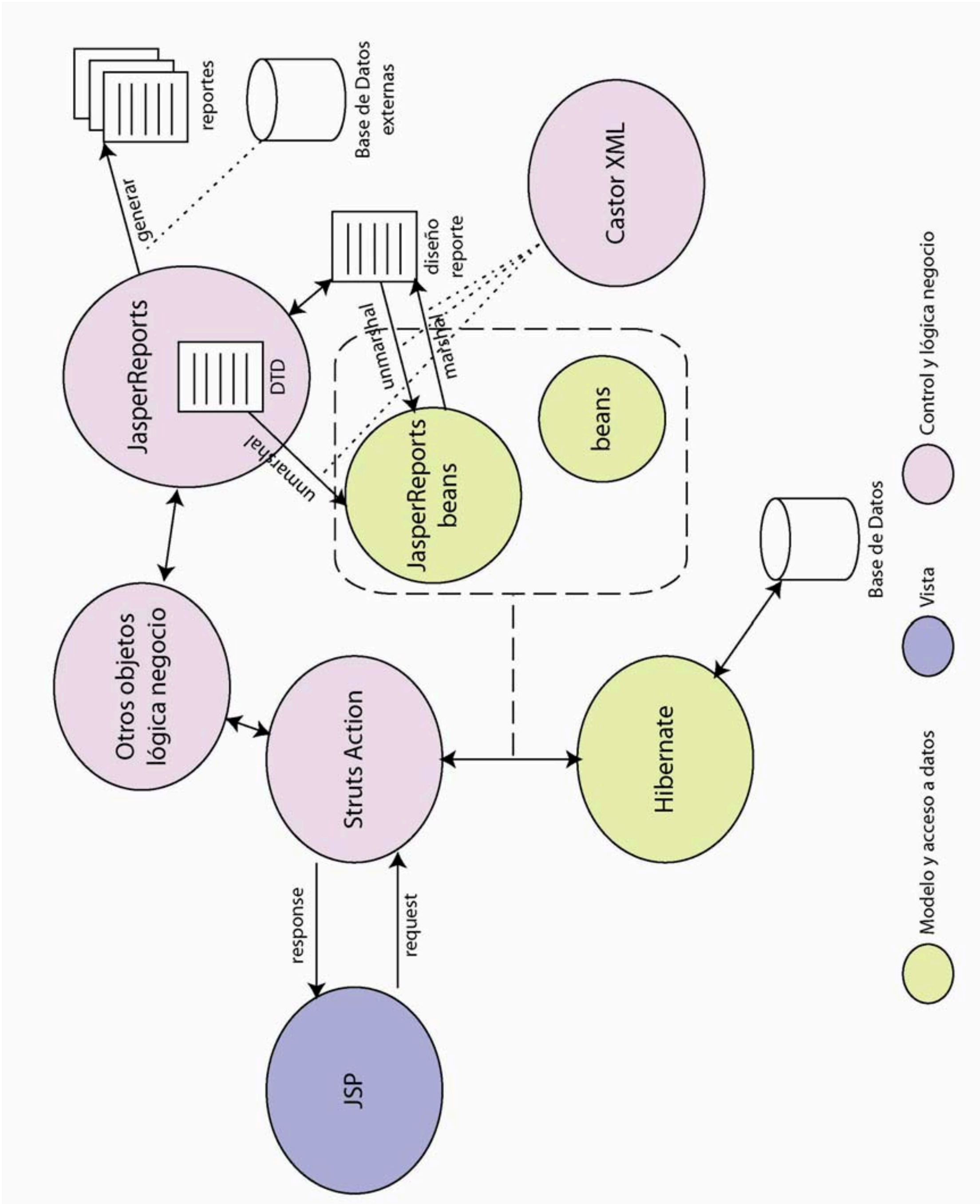


Figura 27: Tecnologías en arquitectura MVC de Eguana Reports

- Los JSP constituyen la parte visual y es la que interactúa con los usuarios.

- Sobre la tecnología JSP funciona JSTL (JSP Standard Tag Library), que encapsula funcionalidad común en aplicaciones JSP. Con los tags de JSTL se pueden crear bucles, condiciones, manipular documentos XML, soportar internacionalización, y otras tareas de programación. Además de JSTL disponemos de los tags de Struts, útiles para crear formas[20] y otros elementos visuales basados en HTML.

- En la capa de control, los objetos que implementan la funcionalidad de Struts (clases Action) procesan las peticiones (request) y envían la respuesta (response) a la capa de vista. Delegan funcionalidad de negocio a los respectivos componentes.

- Por otro lado tenemos los beans. En este caso hemos hecho alusión explícita a JasperReports beans, que son la representación en objetos de la estructura DTD[21] de JasperReports. Esto nos permite la creación dinámica de diseños de reporte. Estos beans se construyen una sóla vez en la etapa de desarrollo de la aplicación. Los otros beans

---

[20] Formas para ingreso de información del usuario.
[21] Document Type Definition, define los elementos permitidos en la construcción de un documento XML.

corresponden a los objetos que analizamos antes en la sección de Funcionalidad.

- Los beans son utilizados por los objetos de la capa de control para realizar las tareas del negocio y se almacenan en las bases de datos haciendo uso de Hibernate. Hibernate nos permite almacenar objetos en bases de datos relacionales y consultarlas luego, convirtiendo los datos en objetos.

- El diseño de reporte (su estructura XML y su contenido) es convertido a objetos para facilitar el diseño dinámico, y a su vez, convierte el diseño dinámico en un documento XML (extensión JRXML). Esta conversión se la hace con Castor XML.

- Castor tiene la función de hacer marshal (convertir objeto a secuencia de bytes) y unmarshal (convertir una secuencia de bytes en objeto). Con los beans de JasperReports, creados una sola vez en momento de programación, se mantiene una relación de uno a uno entre los beans (objetos) y los elementos XML (documento) del diseño y se facilita la implementación de la solución final para el usuario.

- La petición para generar y compilar reportes se hace con JasperReports, que utiliza el diseño y accede a otras bases de datos para adquirir información.

3.6. Módulo de Administración de Reportes

Vamos a describir en esta sección varios aspectos de diseño para implementar los casos de uso de Eguana Reports. Para tener una mejor visión, adoptaremos una perspectiva desde la implementación y mostraremos el prototipo de la Vista (interfaz de usuario).

A nivel de programación los objetos de Control heredan la funcionalidad de `DispatchAction`, que se encuentra en la librería de Struts.

Mostramos a continuación nuestra metodología de trabajo en el diseño de las tareas más significativas de Eguana Reports. Tareas redundantes, tal como "eliminación", no las mencionaremos. Nuestro propósito es ser específicos, por fines educativos.

3.6.1. Creación de grupos

Objetivo

Crear grupos de usuarios que compartan un mismo fin. Lo más común es que los usuarios sean agrupados por empresa y/o departamentos.

Caso de uso

Esta funcionalidad se encuentra dentro del caso de uso 2, Administrar Grupos.

Vista

Para cumplir el objetivo se necesitan las siguientes presentaciones:

- Interfaz para creación de grupos.

Control

El objeto de control `GruposAction` es responsable de responder a los siguientes requerimientos:

- Agregar grupo: el requerimiento para proveer la Vista (JSP) correspondiente.

- Guardar grupo: el requerimiento del administrador de sistema que se delega a la capa de persistencia para almacenar el objeto `Grupo` en la base de datos.

Modelo

- El objeto `Grupo` es el único involucrado.

Implementación

La interfaz es sencilla, como se muestra a continuación:

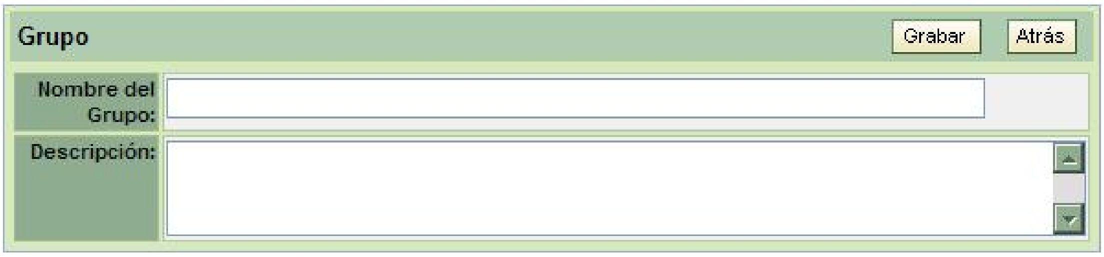


Figura 28: Vista para creación de grupo

Nota Ampliatoria

Cualquier otra tarea con grupos corresponde a modificación del grupo ya creado. El actor, en este caso, es el administrador de sistema.

3.6.2. Creación de usuarios

Objetivo

Crear usuarios que, asociado a un grupo, pueda crear, modificar y obtener reportes. También cumple el objetivo de mantener el control de acceso al sistema, mediante una contraseña privada.

Caso de uso

Esta funcionalidad se encuentra dentro del caso de uso 1, Administrar Usuarios.

Vista

Para cumplir el objetivo se necesitan las siguientes presentaciones:

- Interfaz para creación de usuario.

Control

El objeto de control `UsuariosAction` es responsable de responder a los siguientes requerimientos:

- Agregar usuario: el requerimiento para proveer la Vista (JSP) correspondiente.

- Guardar usuario: el requerimiento del administrador de sistema, que se delega a la capa de persistencia para almacenar el objeto `Usuario` en la base de datos.

Modelo

- El objeto `Usuario` es el único involucrado.

Implementación

La interfaz es sencilla, como se muestra a continuación:

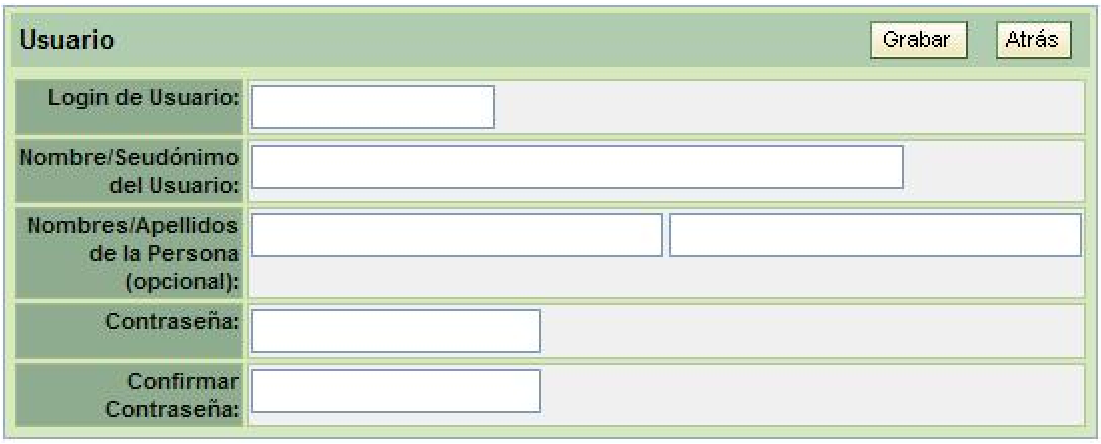


Figura 29: Vista para creación de usuario

Nota Ampliatoria

Cualquier otra tarea con usuarios corresponde a modificación del usuario ya creado. El actor, en este caso, es el administrador de sistema.

3.6.3. Asignación de reportes

Objetivo

Asignar un reporte a un grupo para que los usuarios que pertenezcan a dicho grupo puedan acceder al reporte.

Caso de uso

Esta funcionalidad se encuentra dentro del caso de uso 4, Administrar Reportes.

Vista

Para cumplir el objetivo se necesitan las siguientes presentaciones:

- Interfaz para listar grupos que tienen relación con el reporte.

- Interfaz para relacionar el reporte a un grupo y la política de acceso deseada.

Control

El objeto de control `AdministradorReportesAction` es responsable de responder a los siguientes requerimientos:

- Asignar grupo: el requerimiento del administrador de sistema, que se delega a la capa de persistencia para almacenar el la relación entre el objeto `Reporte` y el objeto `Grupo` en la base de datos.

Modelo

- Los objetos `Reporte` y `Grupo`, que pasan a relacionarse.

Implementación

Mostramos a la interfaz para listar los grupos relacionados con el reporte y su respectiva política de acceso al mismo.

Figura 30: Vista para asignación de reporte

Nota Ampliatoria

Para asignar un reporte primero debe haberse creado. Ver más adelante en Mantenimiento de reportes.

La política por defecto indica que el reporte estará disponible para todos los grupos, si es PERMITIR, o se niega el acceso, si es NEGAR, para luego conceder acceso a grupos específicos. Se puede negar expresamente el acceso (al reporte) a un grupo.

3.6.4. Mantenimiento de grupos

Objetivo

Una vez creado un grupo, se llevan a cabo tareas de modificación para mantener al grupo activo dentro del sistema. Las tareas son de modificación de datos, asignación de usuarios y designación de usuario administrador de grupo.

Caso de uso

Esta funcionalidad se encuentra dentro del caso de uso 2, Administrar Grupos.

Vista

Para cumplir el objetivo se necesitan las siguientes presentaciones:

- Interfaz para buscar y listar grupos del sistema.
- Interfaz para modificar grupo.
- Interfaz para listar usuarios del grupo.
- Interfaz para incluir un usuario al grupo y editar su perfil.

Control

El objeto de control `GruposAction` es responsable de responder a los siguientes requerimientos:

- Buscar y listar grupos: buscar para filtrar la lista global de grupos y escoger el grupo para darle mantenimiento.
- Guardar grupo: que se delega a la capa de persistencia para actualizar la información en la base de datos.

- Listar usuarios del grupo.

- Incluir usuario al grupo: que se delega a la capa de persistencia para almacenar la relación entre el Usuario y el Grupo en la base de datos.

Modelo

- Los objetos `Grupo` y `Usuario`, que pasan a relacionarse.

Implementación

Mostramos a la interfaz para listar los grupos y la interfaz para mantenimiento:

Figura 31: Vista para buscar y listar grupos.

Figura 32: Vista para mantenimiento de grupo.

3.6.5. Mantenimiento de usuarios

Objetivo

Una vez creado un usuario, igual que en grupos, se llevan a cabo tareas de modificación para mantener al usuario activo dentro del sistema. Las tareas son de modificación de datos, asignación de roles y cambio de contraseña.

Caso de uso

Esta funcionalidad se encuentra dentro del caso de uso 1, Administrar Usuarios.

Vista

Para cumplir el objetivo se necesitan las siguientes presentaciones:

- Interfaz para buscar y listar usuarios del sistema.

- Interfaz para modificar usuario.

- Interfaz para listar roles del usuario

- Interfaz para asignar un rol al usuario.

Control

El objeto de control `UsuariosAction` es responsable de responder a los siguientes requerimientos:

- Buscar y listar usuarios: buscar para filtrar la lista global de usuarios y escoger el usuario para darle mantenimiento.

- Guardar usuario: que se delega a la capa de persistencia para actualizar la información en la base de datos.

- Asignar rol al usuario: que se delega a la capa de persistencia para almacenar la relación entre el `Rol` y el `Usuario` en la base de datos.

- Listar roles del usuario.

Modelo

- El objeto `Usuario` es el involucrado.

- El objeto `Rol`, que en realidad no corresponde al modelo del análisis, sino a una solución de diseño e implementación para definir y separar actores del sistema. Además, entra en el ámbito de seguridad del sistema.

Implementación

Mostramos a la interfaz para mantenimiento de usuario. La interfaz para búsqueda y listado es similar a la de los grupos, y no se muestra:

Figura 32: Vista para mantenimiento de usuario

3.6.6. Mantenimiento de reportes

Objetivo

Una vez creado un reporte, igual que en grupos y usuarios, se llevan a cabo tareas de modificación para mantener al reporte vigente dentro del sistema. Y accesible a otros grupos. Las tareas son de modificación, asignación a grupos, además de creación y mantenimiento de parámetros del reporte.

Caso de uso

Esta funcionalidad se encuentra dentro del caso de uso 4, Administrar Reportes.

Vista

La interfaz para creación de reporte es la misma que para modificación. Para cumplir el objetivo se necesitan las siguientes presentaciones:

- Interfaz para buscar y listar reportes del sistema: los reportes resultantes dependen del contexto. Para un usuario aparecerán los reportes del grupo al que pertenece. Para el administrador de sistema aparecerán todos.

- Interfaz para modificar reporte.

- Interfaz para creación y mantenimiento de parámetros del reporte.

- Interfaz para asignación de reportes a grupos.

Control

El objeto de control `AdministradorReportesAction` es responsable de responder a los siguientes requerimientos:

- Buscar y listar reportes: buscar para filtrar la lista de reportes y se puede escoger el reporte para darle mantenimiento.

- Guardar reporte: que delega la validación del diseño a los objetos de lógica de negocio (compilador de Jasper Reports, el archivador en disco, y otros) y a la capa de persistencia la tarea de actualizar la información en la base de datos.

- Asignar reporte a grupo: como se vio en 3.6.3 Asignación de reportes.

- Interfaz para listar parámetros del reporte.

- Interfaz para creación y mantenimiento de parámetros del reporte.

- Interfaz para creación y mantenimiento de valores del parámetro de reporte.

Modelo

- El objeto `Reporte` es el involucrado.

- `Reporte` tiene una lista de parámetros (objeto `Parámetro`). Un `Parámetro` tiene una lista de valores (objeto `ValorParámetro`).

- El objeto `Grupo` que se relaciona con el `Reporte` para conceder acceso (al reporte).

Implementación

Mostramos a la interfaz para mantenimiento de reporte y la interfaz para mantenimiento de parámetros:

Figura 33: Vista para mantenimiento de reporte

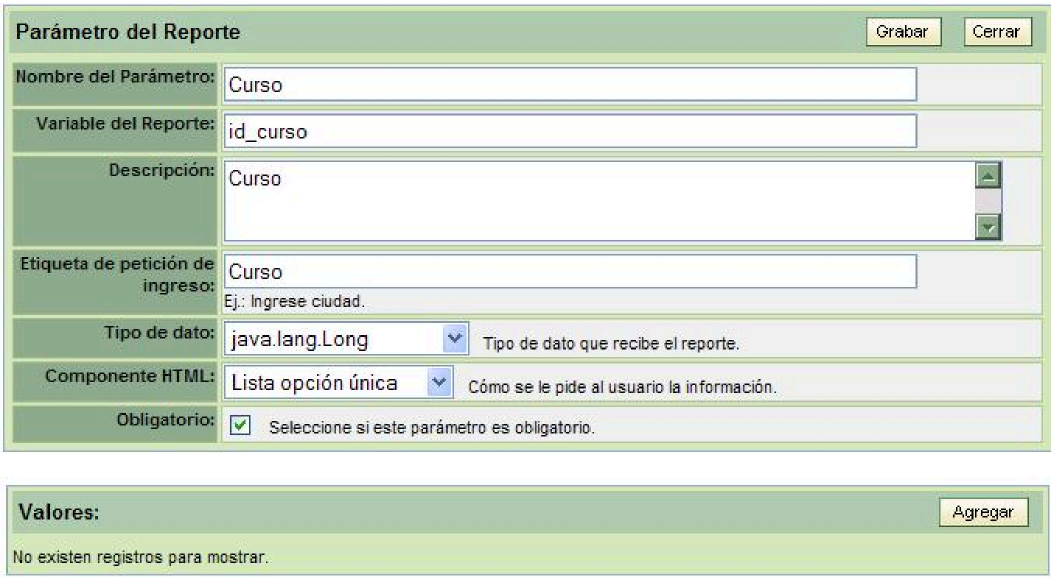


Figura 34: Vista para mantenimiento de parámetro de un reporte

Nota Ampliatoria

Los parámetros representan información o filtros de entrada cuando un usuario va a pedir un reporte. Por ejemplo, un reporte de notas de un curso. Un parámetro de reporte sería el curso para el cual se desea listar las notas. El usuario desea obtener el listado de notas y selecciona el curso “Inglés Avanzado”, y se genera un PDF con las notas de ese curso.

3.7. Roles de los usuarios de Eguana Reports

Los roles corresponden a la etapa de diseño e implementación en Eguana Reports. Se crean ante la necesidad de diferenciar los actores del sistema, además de propósitos de privacidad, seguridad y jerarquía dentro de los usuarios.

Con los roles se define qué parte del sistema ve qué usuario. Así también las opciones del menú del sistema pueden ser configuradas para crear cualquier tipo de rol que combine cualquier funcionalidad disponible en el sistema.

Nosotros creamos un menú de opciones. Luego creamos roles y definimos a qué opciones tendrá acceso el rol. De ahí asignamos al usuario un rol específico.

Las opciones disponibles son:

- Administrar grupos de trabajo.- Administración de grupos de usuarios para el acceso a reportes.

- Administrar reportes.- Control total de los reportes.

- Seguridad.- Creación de usuarios y asignación de roles.

- Datasources.- Administración de fuentes de datos para el uso en los reportes.

- Administrar mis reportes.- Para reportes relacionados al usuario que ingresó al sistema.

- Administrar mis grupos.- Para usuarios que han sido designados administradores de grupo.

- Básico.- Cambio de contraseña y cierre de sesión. También la opción de obtener reportes, que es el propósito de este sistema. Todo usuario tiene acceso a esta opción, pero los reportes que se presenten depende de los permisos que se le concedieron.

Con las opciones definidas creamos 3 roles básicos:

- Administrador del servidor de reportes.

- Administrador de reportes.

- Usuario.

Hemos de recalcar, otra vez, que en este nivel el rol se debe más a una solución de diseño e implementación. Los actores del sistema, aunque sólo 2, se ven reflejados en los roles que hemos definido. Eso NO significa que el rol representa directamente a un actor de nuestros casos de uso.

3.7.1. Administrador del Servidor de Reportes

En nuestros casos de uso es el “Administrador de sistema” y al mismo tiempo es el “Usuario”. Tiene acceso a todo. Puede hacer y deshacer según su criterio. Tiene asignada las opciones:

- Administrar grupos de trabajo.
- Administrar reportes.
- Seguridad.
- Datasources.
- Básico.

Se relaciona con todos los casos de uso. Cuando administra grupos de trabajo, administra reportes, administra usuarios y administra datasources actúa como “Administrador de sistema”. Cuando diseña reportes, obtiene reportes y cambia la clave de acceso actúa como “Usuario”.

3.7.2. Administrador de Reportes

Es un intermedio entre un usuario y un administrador de sistema. Puede decirse que actúa como “Usuario”, pero a la vez como “Administrador de Sistema” con limitaciones.

Hay ciertos usuarios a los que se les asigna como administradores de grupo. Estos podrán administrar su grupo y sus reportes. Su campo de acción se limita a los grupos de los que es administrador.

Las opciones asignadas son:

- Administrar mis reportes.
- Administrar mis grupos.
- Básico.

Cuando administra reportes de su grupo y administra usuarios de su grupo actúa como "Administrador de sistema". Cuando diseña reportes, obtiene reportes y cambia la clave de acceso actúa como "Usuario".

3.7.3. Usuarios

Son los beneficiarios de la función principal de Eguana Reports, la de generar reportes. Un usuario no se ve involucrado en tareas de creación y mantenimiento. Simplemente ingresa al sistema para obtener reportes.

La opción para este caso es:

- Básico.

Aunque suene redundante, alguien con rol Usuario actúa como “Usuario” según nuestro caso de uso. Aquí definimos una relación de uno a uno entre el rol implementado y el actor de los casos de uso.

3.8. Reportes

Los reportes son la razón de ser de nuestro sistema, centrándonos en la herramienta que es la base de Eguana Reports.

Veamos más en detalle cómo funciona JasperReports.

3.8.1. Cómo funciona Jasper Reports

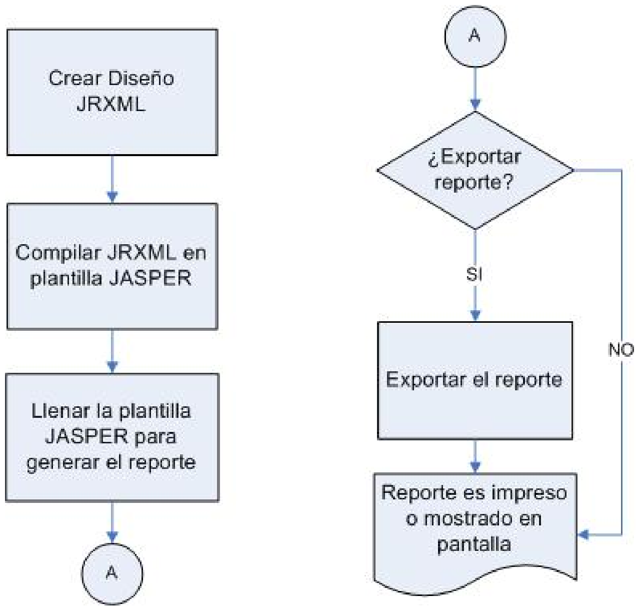


Figura 35: Diagrama de flujo para crear reportes en JasperReports

El primer paso para crear un reporte con JasperReports es crear la plantilla de diseño en un archivo XML. La plantilla XML puede ser escrita a mano o generada por un programa gráfico. A pesar de ser archivos XML, reciben la extensión `JRXML`, y por lo tanto son usualmente referidos como los “archivos JRXML”.

Veamos la estructura de un archivo JRXML con los principales elementos:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report
Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport name="Reporte">
  <title>
    <band height="30">
    </band>
  </title>
  <pageHeader>
    <band height="30">
    </band>
  </pageHeader>
  <columnHeader>
    <band height="30">
    </band>
  </columnHeader>
  <detail>
    <band height="30">
    </band>
  </detail>
  <columnFooter>
    <band height="30">
    </band>
  </columnFooter>
  <pageFooter>
    <band height="50">
    </band>
  </pageFooter>
  <lastPageFooter>
    <band height="50">
    </band>
  </lastPageFooter>
```

```
  <summary>
    <band height="50">
    </band>
  </summary>
</jasperReport>
```

Estos son los elementos principales de un reporte en JasperReports: título, cabecera de página y pie de página, cabecera de columna y pie de columna, detalle, un pie de página final y un sumario.

Todos los elementos contienen el elemento hijo `<band>`, que es una sección transversal dentro de la página del reporte a la que se le define una altura. Dentro de cada “banda” se ponen elementos de texto, imágenes y otros. Existen herramientas de diseño visual en el mercado, aunque la oficial para JasperReports es iReport.

El archivo JRXML es compilado a una plantilla de código nativo de JasperReports, ya sea en momento de programación llamando al método `compileReportToFile()`, o utilizando una tarea de Ant. El archivo resultante se conoce como “archivo Jasper” y tiene la extensión `JASPER`.

El archivo Jasper es usado para generar el reporte, llenándolo con los datos necesarios. A este proceso se lo conoce como “llenar” el reporte.

El archivo JRXML se compila una sola vez, y el archivo Jasper puede ser usado muchas veces para generar reportes.

Los reportes llenados (ya generados) pueden ser almacenados en disco en un formato nativo de JasperReports. A estos archivos se los conoce como “archivos JasperPrint” y pueden ser leídos y vistos sólo con una herramienta específica para ver reportes de JasperReports.

Los archivos JasperPrint pueden ser convertidos a otros formatos como PDF, XLS, DOC, RTF, XML, CSV. En este caso pueden ser leídos por las aplicaciones disponibles en el mercado como Acrobat Reader, Word, Excel. Esta es la opción que utilizamos en nuestro sistema.

3.8.2. Diseño y almacenamiento de reportes

El diseño de un reporte, entonces, sirve para crear el formato visual donde se utilizan textos, fuentes, líneas, cuadrados, colores, imágenes, entre otros, y organizar su disposición en la página con la ayuda de las bandas y los elementos cabecera, sumario, título, y demás.

Además se determina cómo obtiene los datos y qué datos va a usar, así como los parámetros de entrada del reporte. Es en esta parte donde Eguana Reports debe facilitar la tarea.

Existen dos formas de proveer un archivo JRXML al sistema:

La primera es crear el diseño en una herramienta visual, como iReport, y luego cargándolo al sistema con la interfaz para creación y mantenimiento de reportes.

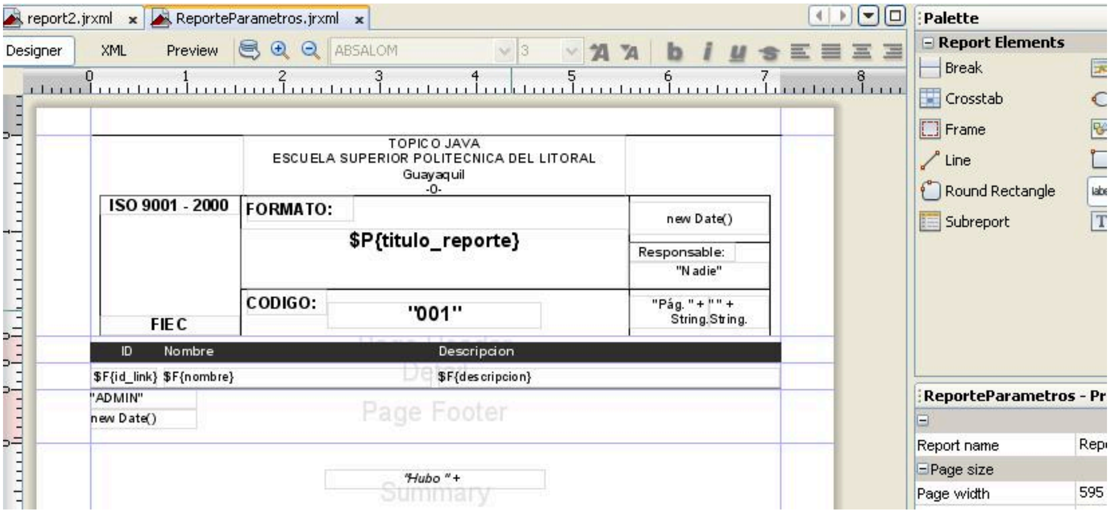


Figura 36: Diseño de reporte con iReport

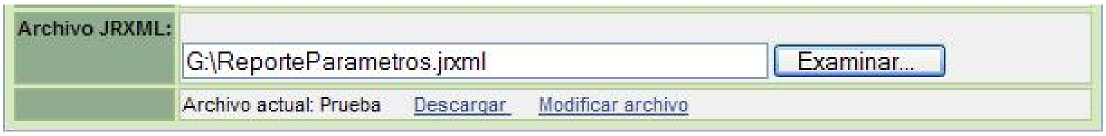


Figura 37: Cargar un archivo JRXML en Eguana Reports.

La segunda es crear el diseño mediante la interfaz de Eguana Reports. Este diseñador no es visual, pero es un prototipo que sirve para generar reportes básicos de una manera muy sencilla y dinámica. El nivel de conocimientos necesarios para diseñar un reporte con Eguana Reports es menor.

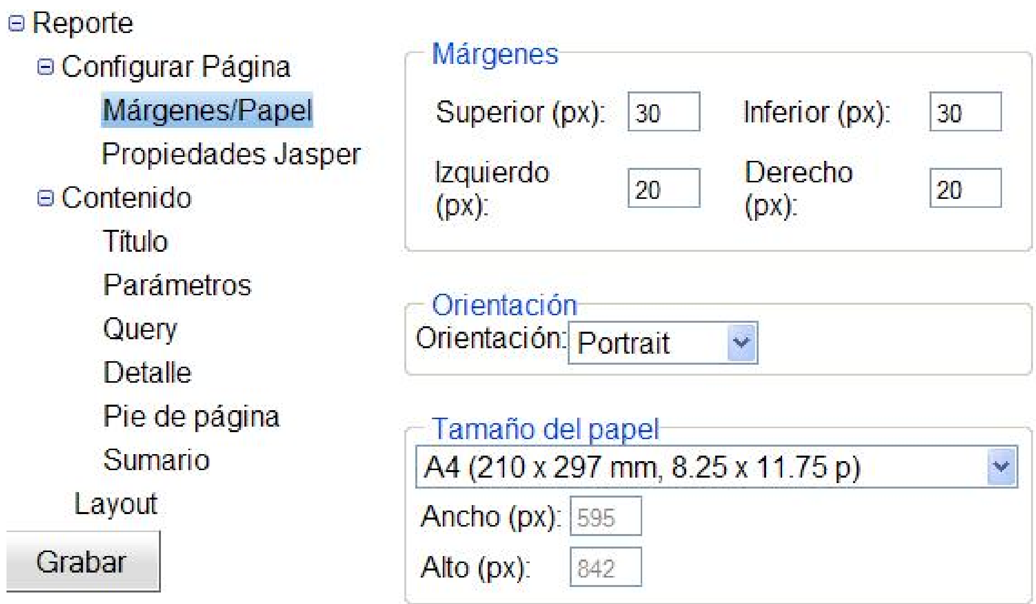


Figura 38: Diseñar un reporte con Eguana Reports

De cualquiera de las dos formas, Eguana Reports se encarga de compilar el archivo JRXML. Si el diseño no es válido entonces no se podrá almacenar.

El almacenamiento de los archivos JRXML y Jasper se hace en disco, en un repositorio expresamente creado para este propósito, dependiendo del grupo propietario:

Figura 39: Archivo JRXML almacenado en disco

El objeto `Reporte`, en nuestro modelo, queda relacionado con el objeto `Archivo`, que hace referencia al archivo en disco JRXML y su compilado, Jasper.

3.8.3. Ejecución de reportes

Todo el proceso para generar un reporte, descrito en 3.8.1, Cómo funciona Jasper Reports, a excepción del diseño, es hecho automáticamente por Eguana Reports.

Objetivo

El usuario desea obtener un reporte. Un archivo Jasper está listo para ser llenado y generar el reporte. La idea aquí es que este proceso sea simplificado, y un usuario pueda obtener el reporte deseado con unos pocos clics.

Caso de uso

Esta funcionalidad se encuentra dentro del caso de uso 6, Obtener Reporte.

Vista

Para la ejecución de un reporte y posterior obtención del mismo se necesitan las siguientes presentaciones:

- Interfaz para buscar y listar reportes: Igual depende del contexto como vimos en el mantenimiento de reportes.

- Interfaz para ingresar parámetros de entrada y definir el formato de salida: XLS, PDF, DOC, y demás disponibles.

- Interfaz final donde el usuario puede ver su reporte o descargarlo.

Control

El objeto de control `ReportesAction` es responsable de responder a los siguientes requerimientos:

- Buscar y listar reportes autorizados: buscar para filtrar la lista de reportes y se puede escoger el reporte para generarlo.

- Entregar la lista de parámetros de entrada que corresponden al reporte.

- Recibir los valores ingresados para los parámetros, el tipo de salida escogido y la solicitud para generar reporte.

Modelo

- El objeto `Reporte`, `Parámetro` y `ValorParámetro` son los involucrados directamente.

- Indirectamente entran en escena `Grupo` y `Usuario`, en la parte de listado de reportes.

Implementación

Mostramos a la interfaz para ingreso de parámetros

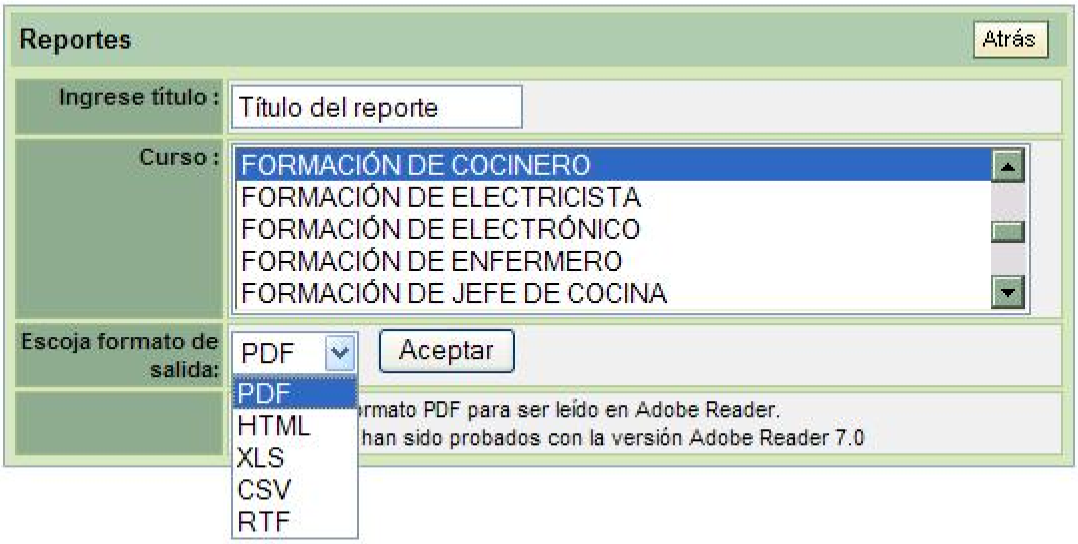


Figura 40: Vista para ingreso de parámetros de reporte

Nota Ampliatoria

`ReportesAction` delega a un objeto de lógica de negocio llamado `Reporteador`, que recibe un objeto `Reporte` y se encarga de todas las tareas involucradas con JasperReports (compilar, acceder a la fuente de datos, llenar, generar y convertir de formato). `Reporteador` es quien interactúa directamente con JasperReports.

3.8.4. Formatos de salida de los reportes

Cuando el archivo Jasper es llenado con los datos se obtiene el archivo JasperPrint, que sólo se puede ver con una herramienta específica para el caso.

El archivo JasperPrint, sin embargo, puede ser convertido a varios formatos que se pueden leer en aplicaciones comerciales, como Adobe Reader (PDF), Microsoft Excel (XLS) y Word (RTF), navegadores web (para HTML), hojas de cálculo y bases de datos (CSV[22]).

El objetivo es darle al usuario muchas posibilidades para elegir el reporte final y abrirlo con su aplicación preferida.

3.8.5. Selección de modelos de reportes y plantillas

Definir el estilo puede ser la diferencia entre un reporte agradable y otro desagradable, a pesar de que posean la misma información. El diseño visual es una parte que a muchos clientes les interesa. Sin embargo no es lo esencial del reporte.

---

[22] Comma Separated Values, o en castellano Valores Separados por Coma, son documentos de formato sencillo para representar datos en forma de tabla y que pueden ser exportados e importados fácilmente por hojas de cálculo y bases de datos.

Eguana Reports provee un estilo básico de reporte, lo necesario para crear una vista aceptable. A través de una plantilla Eguana Reports proporciona la base escalable sobre la cual se pueda aumentar los estilos de diseño. Esto se logra gracias a que el diseño de reportes en Eguana Reports se hace con componentes visuales que representan directamente a los elementos del archivo JRXML. De esa manera lo único que se debe hacer es aumentar otro componente más al diseñador, sin tener que descifrar el código.

El diseño de plantillas se realiza mejor con herramientas visuales, como iReport. El documento básico que genera Eguana Reports puede abrirse perfectamente en iReport y añadirle estilo con mucha más facilidad. Lo único que habría que hacer luego es volverlo a cargar en el servidor.

En fin, significa que nuestro servidor de reportes nos provee una plantilla de diseño sencilla que puede ser mejorada con una herramienta visual.

CAPÍTULO 4

4. Implementación

El proceso de implementación ha sido arduo. Durante el proceso hemos probado varias herramientas, como Sofia, Lomboz, plugins para Eclipse, así como versiones anteriores de Tomcat, JBoss, Eclipse.

Desarrollamos la aplicación haciendo uso de Sofia (Salmon Open Framework for Internet Applications) hasta el punto en que nos dimos cuenta que es inadecuado y limitante para algunos casos. Rehicimos la aplicación usando Struts.

Asimismo analizamos que la tecnología EJB[23] es, además de complicada, muy compleja para nuestros simples propósitos. En vez de eso utilizamos POJO[24] y Hibernate.

Sin los EJB en nuestra implementación, ya no es necesario un contenedor de EJB y lo único que requerimos es un contenedor web, como Apache Tomcat.

---

[23] EJB (Enterprise Java beans). Los beans empresariales son componentes del lado del servidor que encapsulan la lógica de negocio de una aplicación.
[24] POJO (Plain Old Java Object). Son clases simples que encapsulan información y no dependen de un framework especial.

No necesitamos JBoss, aunque la aplicación se ejecuta en este sin problemas. Claro está que Tomcat está embebido en JBoss, pues es su contenedor web por defecto. El aspecto de las transacciones se maneja con Spring Framework, como vimos en la capa de persistencia.

Además, ante el reto de crear un diseñador de reportes lo suficientemente dinámico, nos adentramos en AJAX[25] y creamos un subproyecto con GWT 1.5.2, la herramienta de Google para crear contenido dinámico en las páginas web. A este subproyecto lo llamamos JasperReportDesign y se compila como JavaScript que luego se integra a Eguana Reports.

En los anexos a este documento presentamos detalles de estas otras herramientas, más que nada desde el punto de vista de implementación, cómo instalar, y la bitácora. Creemos interesante, desde el punto de vista educativo, dar a conocer el recorrido hecho hasta llegar al prototipo Eguana Reports final. Se dará el lector cuenta de las dificultades y trabajo a los que nos hemos enfrentado.

Como explicamos en un inicio, el diseño pequeño de nuestra aplicación se ve compensado con la dificultad tecnológica para implementarlo.

A continuación los pasos de instalación y configuración de las herramientas.

---

[25] AJAX *(*Asynchronous JavaScript And XML) (JavaScript Asíncrono y XML)

4.1. Instalación de J2SDK

J2SDK (Java 2 SDK)[26], actualmente J2SE Development Kit, provee un instalador con ayuda que hace fácil la tarea. Además incluye Java RunTime Environment (JRE), necesario para JDK.

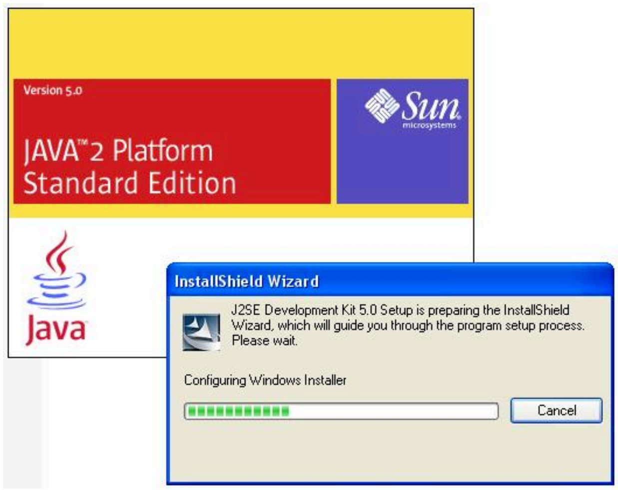


Figura 41: Instalando J2SE Development Kit

1. Se ejecuta el archivo jdk-1_5_0-windows-i586.exe.

2. Se selecciona el directorio de instalación y se sigue la guía del instalador.

3. Se configura las *variables de ambiente del sistema* requerido para Java en el sistema operativo:

% JAVA_HOME% – La ruta base de Java.

% CLASSPATH% – Se definen las rutas del "home" y "bin" de Java.

% PATH% – Se definen las rutas del directorio bin de Java. Agregar "% JAVA_HOME% \bin".

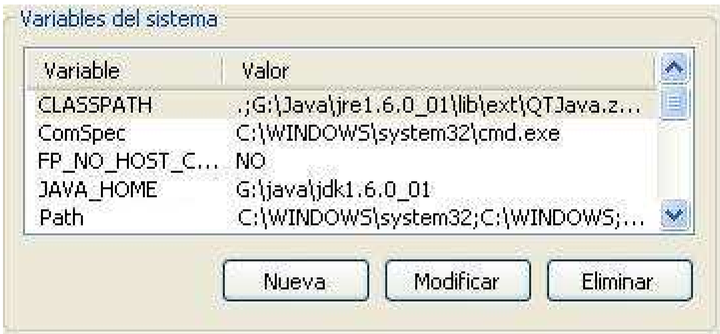


Figura 42: Variables de ambiente para J2SDK

4. Se verifica que la instalación esté hecha correctamente, con la línea de comandos:

```
C:\>java -version
java version "1.6.0_01"
Java(TM) SE Runtime Environment (build 1.6.0_01-b06)
Java HotSpot(TM) Client VM (build 1.6.0_01-b06,
mixed mode, sharing)
```

[26] SDK corresponde a Software Development Kit, aunque actcualmente el nombre es Java 2 SE Development Kit (SE por Standard Edition)

4.2. Instalación del Servidor de Aplicaciones JBoss

JBoss ha resultado ser innecesario y nos limitamos a instalar un contenedor web, que se encuentra en Tomcat 5.5 (JBoss usa Tomcat como contenedor web). En los anexos, sin embargo, se puede encontrar los pasos para instalar JBoss.

1. Se ejecuta el archivo jakarta-tomcat-5.5.9.exe.

2. Se selecciona el directorio de instalación y se sigue la guía del instalador.

3. Se configura las variables de ambientes requeridas para TOMCAT:

   %CATALINA_HOME% – La ruta base donde se instala Tomcat.

   %CLASSPATH% – Se define la ruta del "home" de Tomcat.

4. Tomcat provee un monitor que se puede ejecutar desde el menú de programas. Con el monitor podemos hacer todas las configuraciones, iniciar y parar la ejecución de Tomcat.

5. Verificar que la instalación de Tomcat esté bien hecha. Usando el navegador web ingresar la siguiente dirección: http://localhost:8080 . Debe salir la pantalla de bienvenida de Tomcat.

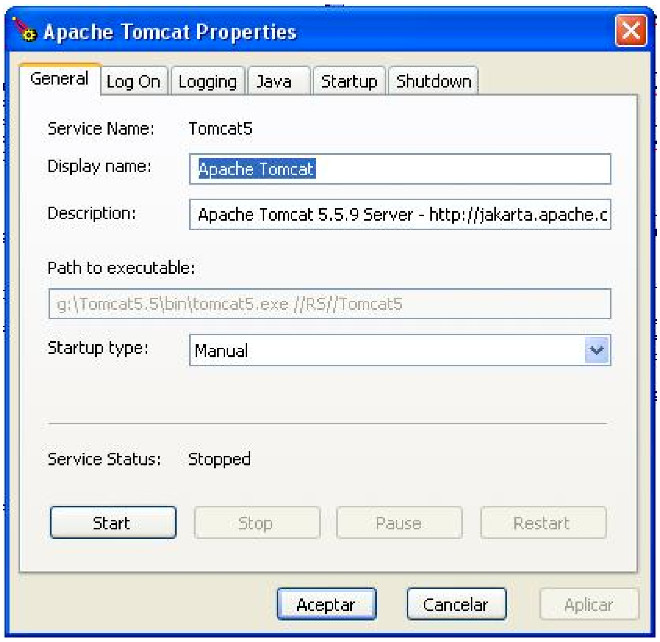


Figura 43: Monitor de Tomcat

4.3. Instalación de la base de datos MySQL

1. Se ejecuta el archivo MySQL-5.1.39-win32.msi

Figura 44: Instalando MySQL

2. Seguir los pasos de la guía de instalación y seleccionar el directorio de instalación.

Figura 45: Guía de instalación para MySQL

3. Verificar la correcta instalación. Se puede iniciar el servicio con el administrador de MySQL o con la línea de comandos. Si se hace con la línea de comandos:

Iniciar y detener como servicio multiusuario:

```
%MYSQL_HOME%\bin:\> net start mysql
%MYSQL_HOME%\bin:\> net stop mysql
```

Iniciar y detener como monousuario:

```
%MYSQL_HOME%\bin:\> mysqld --standalone
%MYSQL_HOME%\bin:\> mysqld-max --standalone
```

```
%MYSQL_HOME%\bin:\> mysqld-max --standalone -debug

%MYSQL_HOME%\bin:\> mysqladmin -u root shutdown
```

4. Levantar una terminal transaccional a través de la línea de comando y verificar el estado:

```
%MYSQL_HOME%\bin:\> mysql
mysql> status;
```

4.4. Implementación de Struts

Para utilizar Struts nos apoyamos con MyEclipse plugin, que nos provee un entorno más agradable para la codificación.

No hay un orden específico para crear los componentes de la estructura con Struts, pero podemos definir lo que se necesita. Para explicar mejor haremos seguimiento a la funcionalidad para administrar usuarios.

Struts tiene un archivo de configuración para inicializar sus propios recursos, semejante a como una aplicación web tiene su archivo web.xml. El archivo de configuración de Struts se llama `struts-config.xml`. En éste se definen las formas y las acciones (las entidades de Control en MVC) que se relacionan con la Vista (JSPs en nuestro caso).

Antes de entrar en detalle, en Eguana Reports usamos Spring, que es una plataforma que se conecta con Struts, Hibernate y que nos permite un mejor control de la seguridad y transacciones.

En `struts-config.xml` se define primero la configuración de Spring:

```
<plug-in
className="org.springframework.web.struts.ContextLoaderPl
ugIn">
   <set-property property="contextConfigLocation"
            value="/WEB-INF/applicationContext.xml,
                   /WEB-INF/action-servlet.xml"/>
</plug-in>
```

Lo que decimos con esto es que dejamos que Spring se encargue del control del contexto de Struts, y se indica los archivos de configuración de Spring. Tener en cuenta `action-servlet.xml`. Para Spring también hay que configurar el contexto en `web.xml`:

```
<listener>
 <listener-class>
  org.springframework.web.context.ContextLoaderListener
 </listener-class>
</listener>
```

Bueno, también se define el validador de formas (formas de ingreso de datos).

```
<plug-in
className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property
```

```
    property="pathnames" value="/WEB-INF/validator-
rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
```

La validación sigue unas reglas (`validator-rules`) y nosotros configuramos cómo validaremos las formas en `validation.xml`.

El siguiente paso es definir qué vistas controla qué acción, y qué forma (y elementos) estará disponible para todas las vistas. Para la parte de usuarios tenemos definidas las páginas JSP siguientes:

```
<action path="/usuarios"
type="org.springframework.web.struts.DelegatingActionProx
y" name="usuarioForm" scope="session" parameter="method"
validate="false">
<forward name="index"
path="/public/jsp/usuario/index.jsp" />
<forward name="administrar"
path="/public/jsp/usuario/administrar.jsp" />
<forward name="listar"
path="/public/jsp/usuario/listar.jsp" />
<forward name="editar"
path="/public/jsp/usuario/editar.jsp" />
<forward name="listarRoles"
path="/public/jsp/usuario/listar_roles.jsp" />
</action>
```

Si se dan cuenta en la definición se utiliza usuarioForm (`name="usuarioForm"`), que se define aparte, en el mismo archivo, así:

```
<form-bean name="usuarioForm"
type="org.apache.struts.validator.DynaValidatorForm">
     <form-property name="entidad"
     type="com.ers.security.model.auth.Usuario"/>
```

```
      <form-property name="criterioBusqueda"
      type="java.lang.String"/>
      <form-property name="cmbCriterioBusqueda"
      type="java.lang.String"/>
      <form-property name="criterioBusquedaRol"
      type="java.lang.String"/>
      <form-property name="cmbCriterioBusquedaRol"
      type="java.lang.String"/>
</form-bean>
```

En `validation.xml` se pone cómo se desea validar la forma, por ejemplo para decir que el nombre es requerido (`depends="required"`):

```
<form name="usuarioForm">
      <field property="entidad.nombreUsuario"
      depends="required">
      <arg position="0" key="usuario.etiqueta.nombre"/>
      </field>
</form>
```

Las propiedades de una forma son fácilmente accesibles programáticamente tanto en los JSP como en las acciones.

Desde una acción (que se ejecuta en el servidor) se puede escribir:

```
DynaActionForm dynaForm = (DynaActionForm) form;
String parametro =
(String)dynaForm.get("criterioBusqueda");
```

Y desde el JSP con un tag de Struts:

```
<html:text property="criterioBusqueda" size="30"/>
```

Bien, volvamos a la configuración de la acción. Tenemos que `path="/usuarios"`. Desde una página JSP se define en la forma que quien toma acción (en el submit) es el control asociado a `path` (`<html:form action="usuarios">`). Al hacer submit se delega al objeto encargado en Spring (un proxy), que, luego de hacer el filtro, busca en el archivo `action-servlet.xml` para saber a qué objeto de Control (de Struts) se delega finalmente. En `action-servlet.xml` tenemos:

```
<bean name="/usuarios"
class="com.ers.web.action.UsuariosAction"
singleton="false">
        …….
</bean>
```

El proceso general se puede resumir así:

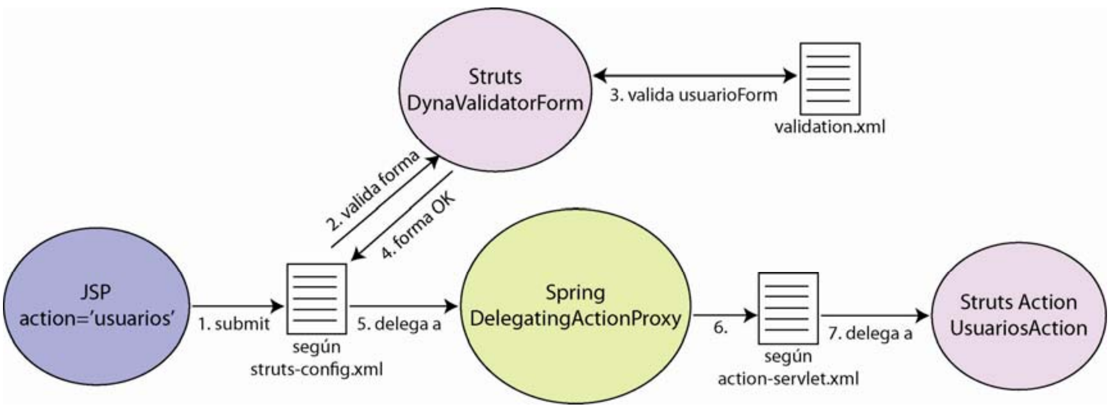


Figura 46: Secuencia de proceso con Struts y Spring

En el lado del servidor, `UsuariosAction` hereda de `DispatchAction` que es una clase de Struts.

```
package com.ers.web.action;
import org.apache.struts.action.DynaActionForm;
import org.apache.struts.actions.DispatchAction;
public class UsuariosAction extends DispatchAction {
public ActionForward buscar(ActionMapping mapping,
ActionForm form,  HttpServletRequest request,
HttpServletResponse response)
    throws Exception {
    DynaActionForm dynaForm = (DynaActionForm)
request.getSession().getAttribute("usuarioForm");
    String parametro =
(String)dynaForm.get("criterioBusqueda");
...
...
    dynaForm.set("entidad", entidad);
    return mapping.findForward("listar");
}
```

Cuando se hace submit a una forma Struts busca el parámetro `method`, que se puede definir en la página JSP, dentro de la forma (form):

```
<html:hidden property="method" value="buscar" />
```

En este caso se llama al método `buscar()` dentro de `UsuariosAction`. De esa forma se puede llamar a cualquier método de cualquier acción desde cualquier JSP. Basta con ubicar bien el código.

El método recibe 4 objetos:

- `mapping`: con el cual se elige la vista a presentar. El nombre de la vista corresponde al que está en `struts-config.xml (<forward name="listar"...>)`.

- `form`: la forma definida en struts-config.xml

- `request`: el objeto HttpServletRequest

- `response`: el objeto HttpServletResponse

La respuesta hacia la interfaz de usuario (Vista) puede ir dentro de la misma forma o contenida en el objeto `response`.

Esos son los principales pasos para implementar Struts. El resto son detalles secundarios.

4.5. Implementación de Hibernate

Hibernate provee de herramientas para convertir los archivos de mapeo a objetos (POJO), pero no hemos hecho uso de esa herramienta.

En cambio, nosotros creamos directamente (“a mano”) los POJO y luego nos valemos de XDoclet y MyEclipse para generar los archivos de mapeo. Primero debemos configurar MyEclipse-XDoclet.

- Si es necesario, reemplazar las librerías de XDoclet del Eclipse para que soporte la versión 3.0 de Hibernate. Crear un archivo `.xdoclet` y revisar que la nueva versión de xdoclet esté en el directorio que dice `xdoclet.basedir` en el archivo `.xdoclet`:

  ```
  C:\Desarrollo\MyEclipse\eclipse\plugins\com.genuitec
  .jboss.ide.eclipse.xdoclet.core_3.9.210
  ```

- Luego Window → Preferences → MyEclipse → XDoclet presionar REFRESH. Y en Code Assist presionar REFRESH.

- Clic derecho sobre el proyecto (EguanaReportsStruts) -> Propiedades.

- Aparece la ventana de propiedades y sobre la derecha seleccionamos MyEclipse-XDoclet. Ahí, en el tab de Configuración habilitamos la opción “Standard Hibernate”.

- Aparecen entonces las propiedades configurables. La que más nos interesa es `destDir`. Es el directorio destino para los archivos de mapeo generados `(.hbm.xml)`, y se encuentra en la ruta base del proyecto. Aquí, con el misma estructura que los paquetes de los POJO, se guardarán los archivos. Nosotros llamamos `destDir=src_cfg2`.

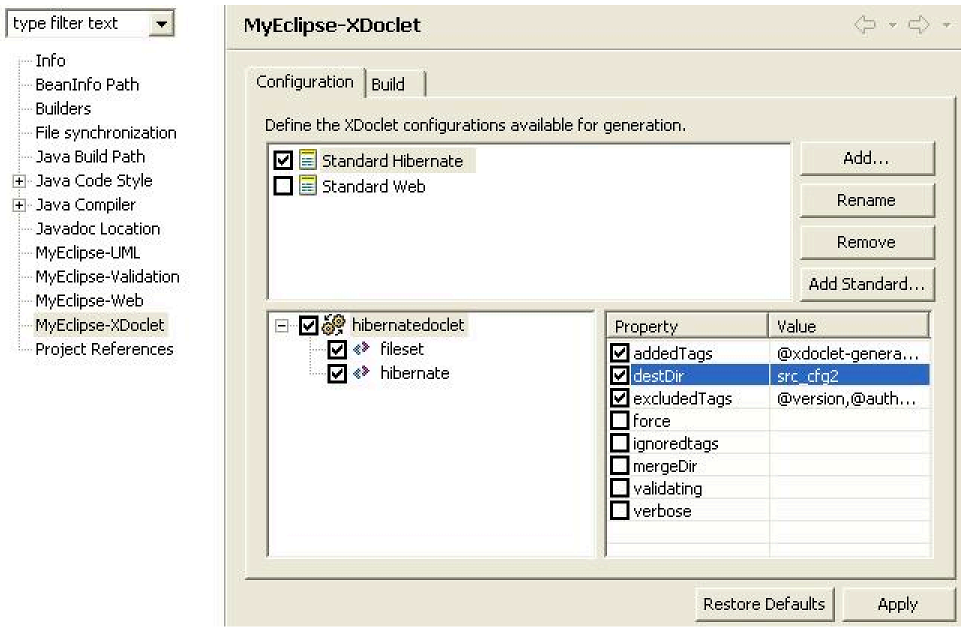


Figura 47: Configuración deMyEclipse-XDoclet para Hibernate

En el POJO agregamos los tags de XDoclet escribiendo justo encima de la declaración de la clase y encima de los métodos `get()` de los atributos que queremos mapear hacia la base de datos. Se usa el identificador `@hibernate`:

```
/**
 * @hibernate.class table="ER_USUARIOS"
 */
public class UsuarioEguana {
     private String userLogin;
     /**
      * @hibernate.id column="user_login" generator-
class="foreign"
      * @hibernate.generator-param name="property"
value="usuario"
      */
    public String getUserLogin() {
        return this.userLogin;
```

```
    }
     /**
     * @hibernate.property
     * @return
     */
    public String getEmail(){
         return email;
    }
...
...
}
```

Para generar los archivos de mapeo hacemos clic derecho en el proyecto y luego MyEclipse -> Run XDoclet. Los archivos `.hbm.xml` estarán en su respectiva ubicación dentro del directorio `src_cfg2`.

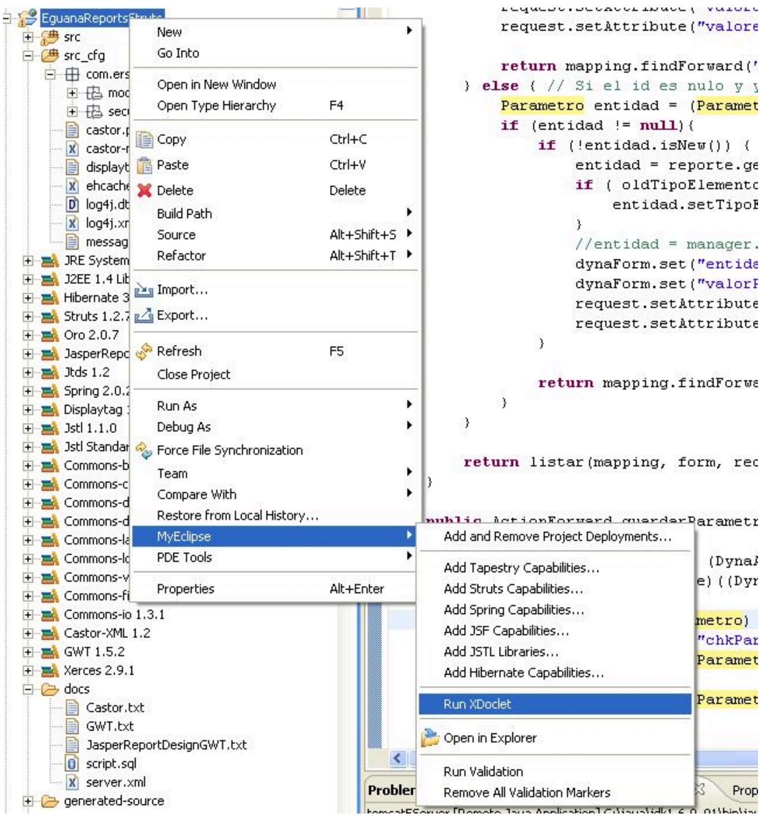


Figura 48: Generar archivos de mapeo de Hibernate

Lo que obtenemos es el archivo de mapeo, aunque igual en casos complejos hay que revisar el código generado para mapear de la mejor manera. Hay veces en que el mapeo automático no resulta y se debe corregir, pero en general vale la pena.

```
<class name="com.ers.model.general.UsuarioEguana"
       table="ER_USUARIOS">

        <id
            name="userLogin"
            column="user_login"
            type="java.lang.String"
        >
            <generator class="assigned">
            </generator>
        </id>

        <property
            name="email"
            type="java.lang.String"
            column="email"
        />
...
...
</class>
```

HIBERNATE EN TOMCAT 5.0

Se debe seguir los siguientes pasos:

- Copiar hibernate3.jar y mysql-connector-java-xxx.jar al classpath del proyecto. Se puede hacer esto haciendo clic derecho en el proyecto

EguanaReportsStruts -> Propiedades -> Java Build Path -> Librerías. En el paquete final de la aplicación deben estar en `WEB-INF/lib`.

- Además de esas librerías, hay que copiar todas las librerías de las que dependa Hibernate para su funcionamiento (leer archivo `Readme` en el directorio `lib` de la distribución de Hibernate):

- dom4j-1.6.1.jar: para hacer parsing[27] de los xml de configuración.

- cglib-full-2.0.1.jar.

- commons-collections-2.1.jar: utiliza varias utilidades.

- odmg-3.0: para hacer mapeo de colecciones.

- ehcache-1.2.3: es del proveedor de caché por defecto si no se cambia en la configuración.

- log4j-1.2.8.jar: commons-logging utiliza log4j si lo encuentra en classpath. Si se va a generar archivo log. Hay que crear el archivo log4j.properties si se utiliza log4j.

---

[27] Parsing: proceso de analizar sintácticamente un texto para determinar su estructura con respecto a una gramática o regla definida.

- commons-logging-1.0.4.jar: una capa (interfaz) para el uso de varias herramientas de log. En este caso se utiliza log4j, para registrar información log y debug.

- commons-logging busca a log4j en el classpath y lo utiliza si lo encuentra. De esta forma no hay que hacer ninguna configuración básica (sólo poner el jar en classpath). Si no encuentra log4j, usa jdk14Logger.

- commons-dbcp-1.2.1.jar: DB Connection Pooling (opcional). Cola de conexión a base de datos. Tomcat ya ofrece mecanismo de pooling.

- commons-lang-2.1.jar: provee funcionalidad extra para las librerías estándares de java (`java.lang` API): métodos para manipulación de cadenas, números, mejoras a `java.util.Date`, entre otros.

Hay que tomar en cuenta que Hibernate usa Java APIs: JNDI, JDBC y JTA.

JNDI y JTA permiten que Hibernate se integre con los servidores de aplicación de J2EE.

HIBERNATE EN SPRING

En el archivo de configuración de Spring (`applicationContext.xml`) se define, a manera de bean, la conexión a la base de datos y los archivos de mapeo, así como otras propiedades de Hibernate:

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerD
ataSource">
  <property name="driverClassName">
     <value>com.mysql.jdbc.Driver</value>
  </property>
  <property name="url">
     <value>
     jdbc:mysql://127.0.0.1:3306/eguanaReports;useCursors
     =true
     </value>
  </property>
  <property name="username"><value>sa</value></property>
  <property name="password">
     <value>topico</value>
  </property>
</bean>
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFac
toryBean">
  <property name="dataSource">
     <ref local="dataSource"/>
  </property>
  <property name="mappingResources">
    <list>
      <value>com/ers/.../Archivo.hbm.xml</value>
      <value>com/ers/.../Datasource.hbm.xml</value>
      <value>com/ers/.../Grupo.hbm.xml</value>
      <value>com/ers/.../Parametro.hbm.xml</value>
      <value>com/ers/.../Reporte.hbm.xml</value>
     ......
    </list>
  </property>
  ......
</bean>
```

4.5.1. Ejemplos de mapeo

Mapeo de atributo a clave primaria en la tabla

En el caso de autogenerar la clave primaria (con el generador de Hibernate):

```
Archivo.java:  protected Long id;
Archivo.hbm.xml:
        <id
            name="id"
            column="id_archivo"
            type="java.lang.Long"
        >
            <generator class="hilo">
                <param name="table">
                    ER_CLAVES_PRIMARIAS
                </param>
                <param name="column">id_archivo</param>
                <param name="max_lo">10</param>
            </generator>
        </id>
```

En el caso de asignar manualmente la clave, como el login de usuario:

```
UsuarioEguana.java: private String userLogin;
UsuarioEguana.hbm.xml:

       <id
            name="userLogin"
            column="user_login"
            type="java.lang.String"
        >
            <generator class="assigned">
            </generator>
        </id>
```

Mapeo de un atributo cualquiera, tipo texto, que puede modificarse, pero no puede ser nulo.

```
Archivo.java: private String nombre;
Archivo.hbm.xml:

        <property
            name="nombre"
            type="java.lang.String"
            update="true"
            insert="true"
            column="nombre"
            not-null="true"
        />
```

Mapeo de uno a muchos (one-to-many).

En el caso de querer una lista ordenada, como en `Reporte`, que se desea tener una lista de Parámetros ordenados para presentar luego en pantalla.

```
Reporte.java: private List<Parametro> parametrosElegibles
              = new ArrayList<Parametro>();
Reporte.hbm.xml:
        <list name="parametrosElegibles"
            table="ER_PARAMETROS"
            lazy="true"
            cascade="all-delete-orphan"
        >
            <key column="id_reporte">
            </key>

            <index
                column="orden"
                type="integer"
            />

            <one-to-many
                  class="com.ers.model.general.Parametro"
            />
        </list>
```

En otro caso, `Grupo` y `Reporte` tienen una relación de muchos a muchos (un grupo puede acceder a muchos reportes y el reporte puede ser accedido por muchos grupos) y se modela el objeto asociación `ReporteGrupo`. La relación entre `Grupo` y `ReporteGrupo` es de uno a muchos. El orden no importa.

```
Grupo.java: private Set reportesGrupos = new HashSet();
Grupo.hbm.xml:
        <set
            name="reportesGrupos"
            table="ER_REPORTES_GRUPOS"
            lazy="true"
            cascade="all"
            sort="unsorted"
        >
            <key column="id_grupo">
            </key>
            <one-to-many
               class="com.ers.model.general.ReporteGrupo"
            />
        </set>
```

Mapeo de muchos a uno (many-to-one)

Siguiendo con el caso anterior, del lado de `ReporteGrupo` se modela la relación de muchos a uno entre éste y `Grupo`.

```
ReporteGrupo.java: private Grupo grupo;
ReporteGrupo.hbm.xml:
        <many-to-one
            name="grupo"
            class="com.ers.model.general.Grupo"
            cascade="none"
            outer-join="auto"
            column="id_grupo"
            not-null="true"
        />
```

Mapeo de uno a uno (one-to-one)

El caso de `UsuarioEguana` (el usuario del servidor de reportes) y Usuario (utilizado en implementación para seguridad), que representan al mismo usuario, pero se implementó en dos para separar funcionalidad.

```
UsuarioEguana.java: private Usuario usuario;
UsuarioEguana.hbm.xml:
        <one-to-one
            name="usuario"
            class="com.ers.security.model.auth.Usuario"
            cascade="none"
            outer-join="auto"
            constrained="true"
        />
```

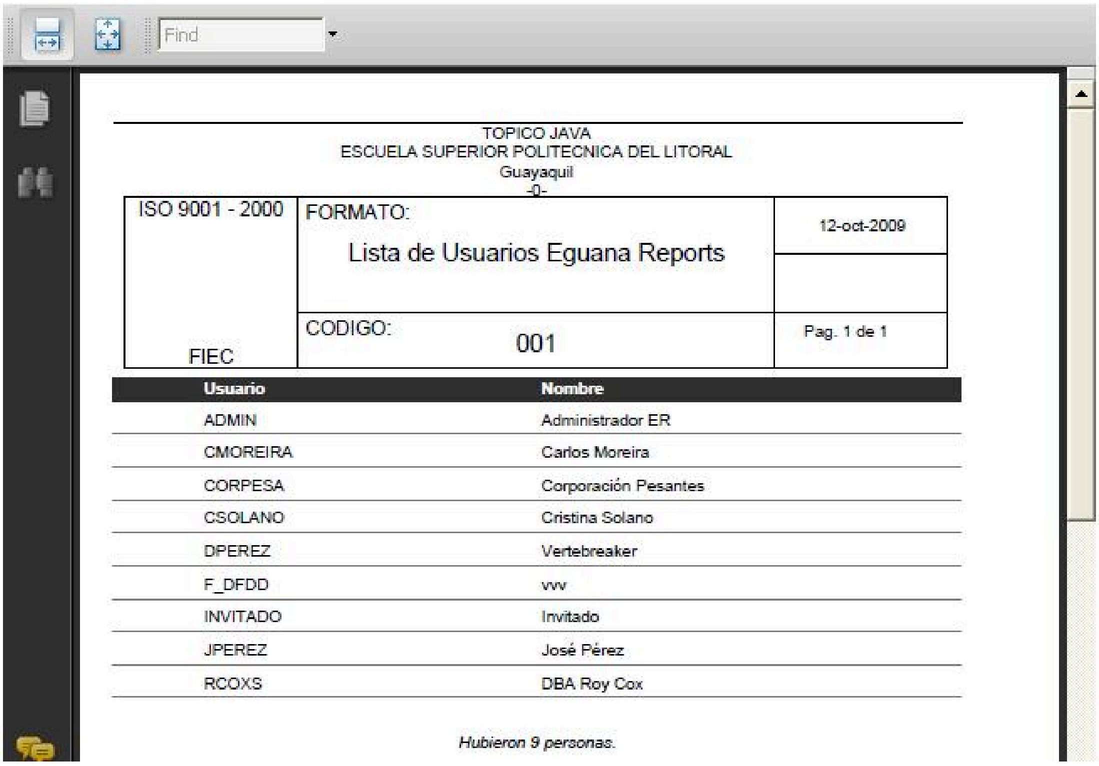
Para más detalles consultar el anexo “Hibernate en detalle”.

4.6. Configuración de JasperReports

En realidad JasperReports es un conjunto de librerías lista para usar. La única configuración necesaria es incluir las otras librerías de las que depende: `itext-1.3.jar`, `poi-2.0-final.jar`, `xml-apis.jar`, `commons-digester-1.6.jar`, `commons-collections-3.1.jar`, `commons-beanutils-1.7.jar`, `commons-logging-1.0.4.jar`.

4.6.1. Ejemplos de reportes

Vamos a poner el ejemplo de un reporte que fue generado con Eguana Reports y luego estilizado para tomar un mejor aspecto.



Find

TOPICO JAVA
ESCUELA SUPERIOR POLITECNICA DEL LITORAL
Guayaquil
-0-

ISO 9001 - 2000	FORMATO: Lista de Usuarios Eguana Reports	12-oct-2009
FIEC	CODIGO: 001	Pag. 1 de 1

Usuario	Nombre
ADMIN	Administrador ER
CMOREIRA	Carlos Moreira
CORPESA	Corporación Pesantes
CSOLANO	Cristina Solano
DPEREZ	Vertebreaker
F_DFDD	vvv
INVITADO	Invitado
JPEREZ	José Pérez
RCOXS	DBA Roy Cox

Hubieron 9 personas.

Figura 49: Ejemplo de reporte

Este reporte se divide en 5 secciones: título, cabecera de página (la banda negra con el nombre de la columna), detalle (la información de los usuarios), pie de página (al final de cada página) y sumario (donde se hace un conteo "Hubo 9 personas").

El reporte es bastante simple, pero visualmente ha sido estilizado. Cada elemento representa un elemento XML son sus respectivos subelementos. Si miramos en un editor de texto el archivo JRXML, veremos la división por bandas de las secciones y los elementos en cada una. Por motivos de espacio exponemos un vistazo reducido al archivo JRXML, con puntos suspensivos para indicar que hay más:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jasperReport ...>
<jasperReport name="Reporte143" ...>
      <reportFont name="Arial_Normal" ... />
<queryString><![CDATA[SELECT ER_AUTH_USUARIOS....]]></queryString>
<field name="campo_1_" class="java.lang.String" ></field>
<field name="campo_3_" class="java.lang.String" ></field>
<title>
<band height="150">
      <rectangle>
            <reportElement x="7" y="45" width="500" height="105"/>
      </rectangle>
      <line>
            <reportElement x="112" y="115" width="395" height="1"/>
      </line>
      <staticText>
            <reportElement x="114" y="0" width="286" height="47"/>
            <textElement textAlignment="Center">
                  <font isBold="true"/>
            </textElement>
            <text><![CDATA[TOPICO JAVA
                  ESCUELA SUPERIOR POLITECNICA DEL LITORAL
                  Guayaquil
                  -0-]]>
            </text>
      </staticText>
</band>
</title>
<pageHeader>
<band height="20">
      <rectangle>............ </rectangle>
      <staticText>
            .........
            <text><![CDATA[Usuario]]></text>
      </staticText>
      <staticText>
            .........
            <text><![CDATA[Nombre]]></text>
```

```
      </staticText>
</band>
</pageHeader>

<detail>
<band height="20">
      <textField .........>
            <reportElement x="55" y="4" width="200" height="15"/>
            <textFieldExpression class="java.lang.String">
                  <![CDATA[$F{campo_1_}]]>
            </textFieldExpression>
      </textField>
      <textField .........>
            <reportElement x="260" y="4" width="255" height="15"/>
            <textFieldExpression class="java.lang.String">
                  <![CDATA[$F{campo_3_}]]>
            </textFieldExpression>
      </textField>
      ............
</band>
</detail>
<pageFooter>
      <band height="40">
            ............
      </band>
</pageFooter>
<summary>
      <band height="80">
            ............
      </band>
</summary>
</jasperReport>
```

4.7. Configuración de Castor

Existe Castor-XML y Castor-JDO (Java Databinding Objects).

Castor-XML nos sirve para convertir objetos a XML y Castor-JDO para relacionar objetos con la base de datos. Castor-JDO no necesitamos.

Las librerías que debemos incluir en el proyecto son: `castor-1.2.jar`, `castor-1.2-xml.jar`, `xerces-2.9.1.jar`.

Agregar las librerías al proyecto.

Cambiar la compatibilidad del compilador Java en Eclipse. Proyecto Eguana Reports (clic derecho) -> Properties -> Java Compiler -> Compiler compliance level a 5.0.

Crear `castor.properties` en `WEB-INF/` y poner la siguiente línea:

```
org.exolab.castor.xml.naming=mixed
```

Lo anterior es para que los tags salgan `<conLaPrimeraLetraCapital>`

Crear el archivo de mapeo `WEB-INF/mapping.xml`, en donde, más que nada en nuestro caso, definimos cómo se debe tratar las colecciones al convertir a XML y viceversa. Por ejemplo:

```
<mapping>
 <class name="com.ers.model.sql.querySpecification.ClauseFrom">
  <field name="tableReferences"
type="com.ers.model.sql.tableReference.TableReference"
collection="arraylist">
   <bind-xml name="tableReference"></bind-xml>
  </field>
 </class>
 <class
name="com.ers.model.sql.querySpecification.SelectSublistList">
  <field name="selectSublists"
type="com.ers.model.sql.querySpecification.SelectSublist"
collection="arraylist">
   <bind-xml name="selectSublist"></bind-xml>
  </field>
```

```
</class>
<class name="com.ers.model.jasper.TextFieldExpression">
 <field name="clazz"
type="com.ers.model.jasper.types.TextFieldExpressionClassType">
   <bind-xml name="class" node="attribute"></bind-xml>
  </field>
 </class>
</mapping>
```

Para generar el código fuente a partir de un esquema (`.xsd`), agregar a la librería: `castor-1.2-codegen.jar`, `castor-1.2-anttasks.jar`. El esquema se puede convertir a partir del DTD con un software como XMLSpy.

Además, para la generación de los beans (`*.java`), que se hace en momento de desarrollo por una sola vez, se deben incluir las siguientes librerías (ir a Window->Preferences->Ant->Runtime->Classpath): `castor-1.2.jar`, `castor-1.2-xml.jar`, `castor-1.2-codegen.jar`, `castor-1.2-anttasks.jar`, `castor-1.2-xml-schema.jar`, `commons-logging-1.0.4.jar`, `velocity-1.5.jar`.

Intentando ejecutar el Generador de Código desde la línea de comando obtenemos:

```
G:\Proyectos\workspace\EguanaReportsStruts\src\com\ers\
model\jasper>java -cp G:\
Proyectos\lib\castor\1.2\castor-1.2-
codegen.jar;G:\Proyectos\lib\castor\1.2\cast
or-1.2.jar;G:\Proyectos\lib\castor\1.2\castor-1.2-
xml.jar;G:\Proyectos\lib\casto
```

```
r\1.2\castor-1.2-xml-schema.jar
org.exolab.castor.builder.SourceGeneratorMain -i
 jasperreport.xsd -verbose
```

También hay que construir un archivo de Ant con la tarea para generar el código. Viene un ejemplo con la distribución de Castor. Escribir el `build.xml`:

```
<?xml version="1.0" encoding="UTF-8" ?>

<project name="CastorJasperReports" default=""
basedir=".">

<target name="castor:gen:src" description="Generate Java
source files from XSD.">
     <taskdef name="castor-srcgen"

     classname="org.castor.anttask.CastorCodeGenTask"/>
       <mkdir dir="generated" />
       <castor-srcgen file="./jasperreport.xsd"
        todir="generated-source"
        package="com.ers.model.jasper"
        types="j2"
        warnings="true" />
</target>
</project>
```

Hacer clic derecho en el archivo `build.xml` → Run As... → Ant Build.

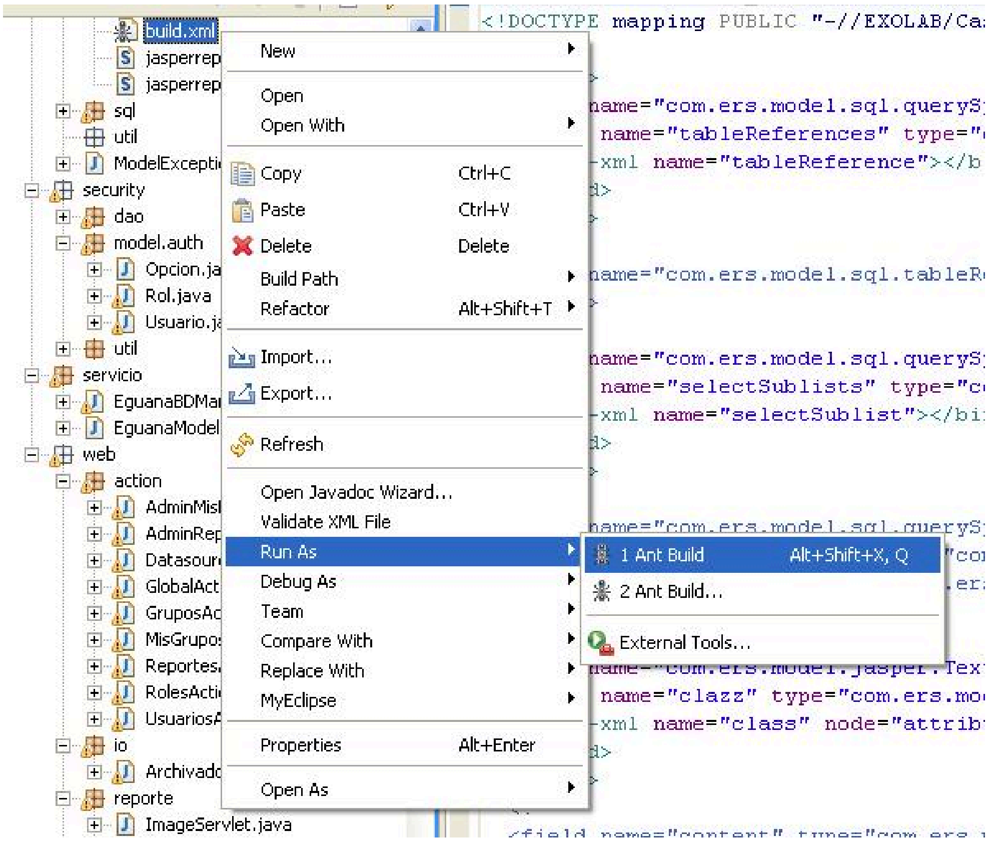


Figura 50: Ejecutando la tarea de Ant para Castor

Y el código se genera en la raíz del proyecto, en el directorio "`generated-source`", como se especifica en `build.xml`.

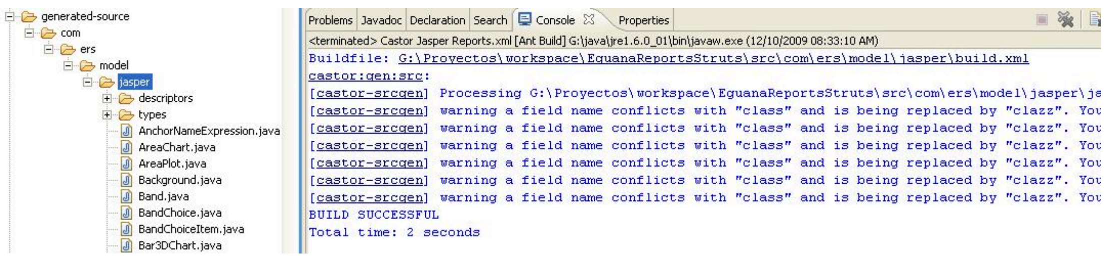


Figura 51: Generar objetos con Castor usando tarea de Ant

4.8. Instalación de Eguana Reports

Para levantar Eguana Reports debemos primero crear las tablas que se muestran en el anexo 1. Por motivos de espacio no incluiremos el código SQL para crear las tablas. La conexión a la base de datos y los archivos de mapeo de Hibernate, como vimos, se configuran en el archivo de Spring, `application-context.xml`.

Debemos asegurarnos que las librerías estén todas en el proyecto.

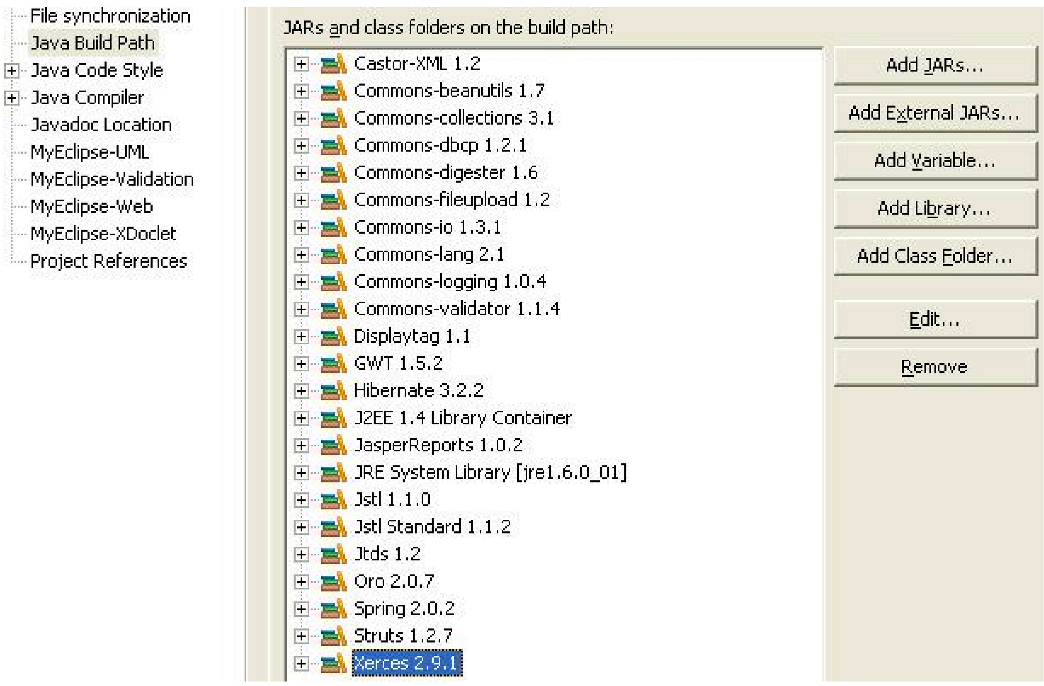


Figura 52: Librerías de Eguana Reports

Configuramos MyEclipse-Web:

- Clic derecho sobre el proyecto EguanaReportsStruts.

- MyEclipse web: Web Context Root: / EguanaReportsStruts.

- En la pestaña Deployment marcar:

    - Use workbench default settings.

    - Jars in web project´s user libraries.

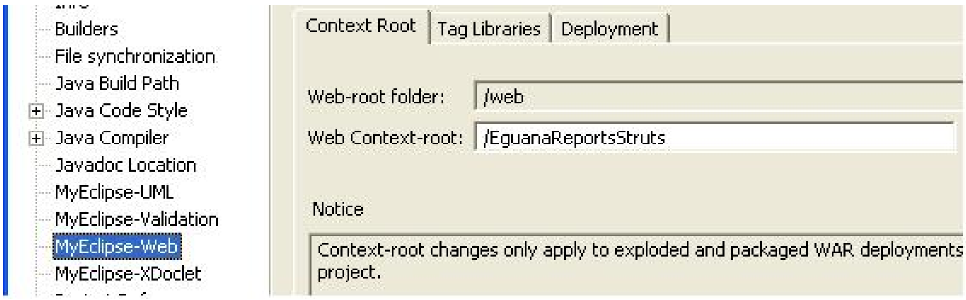


Figura 53: Configuración de MyEclipse-Web

Creamos el war y lo deplegamos directamente en Tomcat:

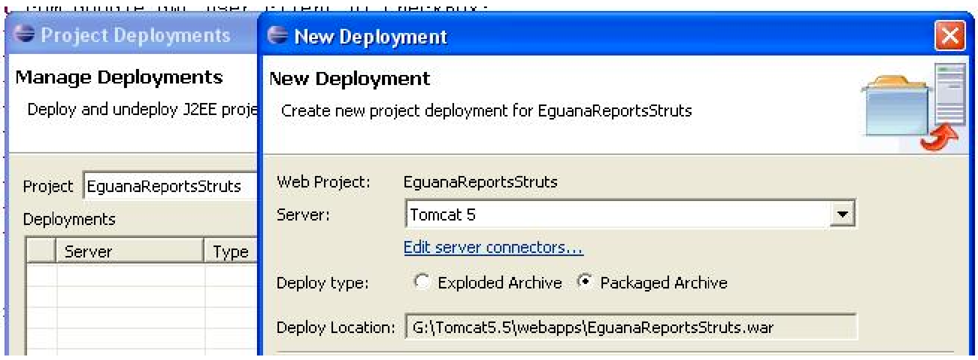


Figura 54: Creación de .war y despliegue en Tomcat

CAPÍTULO 5

5. Plan de pruebas

Aquí presentamos una lista de casos de pruebas básicas que deben hacerse para validar que la aplicación cumpla su objetivo.

Crear un usuario

Crearemos un usuario que se llame “Carlos Moreira”. Para eso ingresamos como Adminsitrador de sistema y en la sección de usuarios creamos el registro. El resultado debe ser:

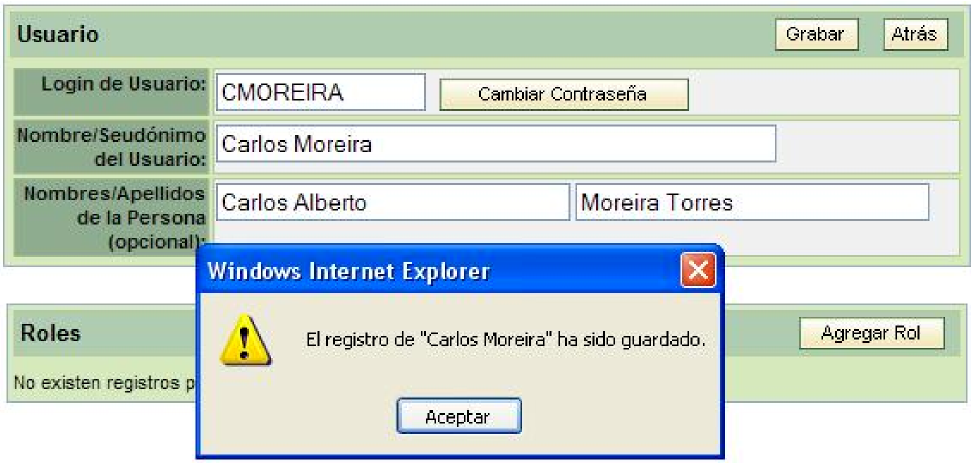


Figura 55: Prueba de creación de usuario

Asignar rol a un usuario

Cuando se crea un usuario aparece en la parte inferior la lista de roles asignados. Al inicio no tiene rol. Con el botón “Agregar Rol” le asignamos el rol básico.

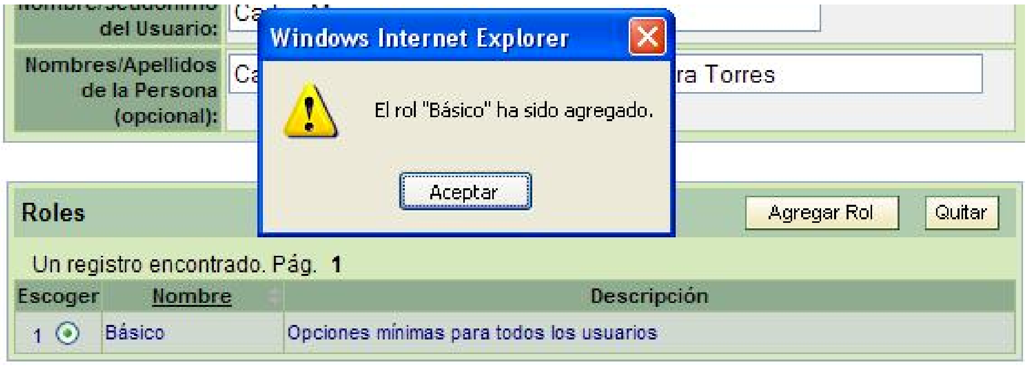


Figura 56: Prueba de asignación de rol a un usuario

Crear un grupo

Crearemos un grupo que se llame “Reportes Básicos”. En la opción de grupos, como Administrador de sistema, creamos el nuevo grupo.

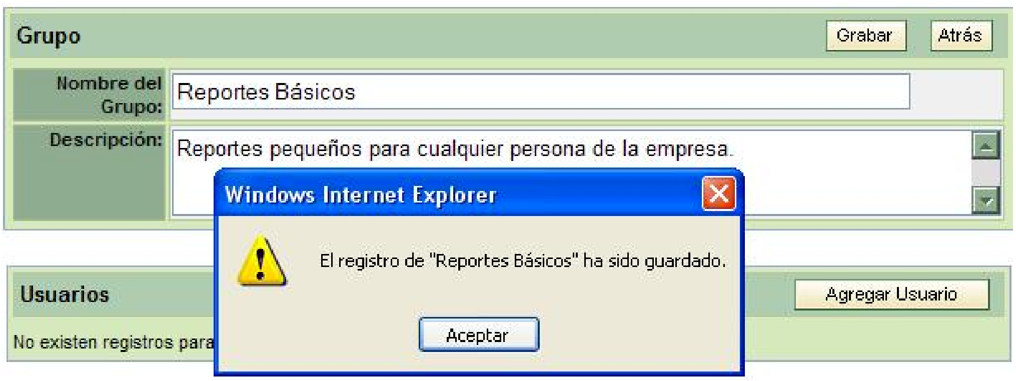


Figura 57: Prueba de creación de grupo

Incluir a un usuario en un grupo

Vamos a incluir a Carlos Moreira en el grupo de Reportes Básicos. Cuando se crea el grupo, en la parte inferior aparece una lista de usuarios que pertenecen al grupo. Al inicio no hay nadie. Con el botón “Agregar Usuario” incluimos a Moreira:

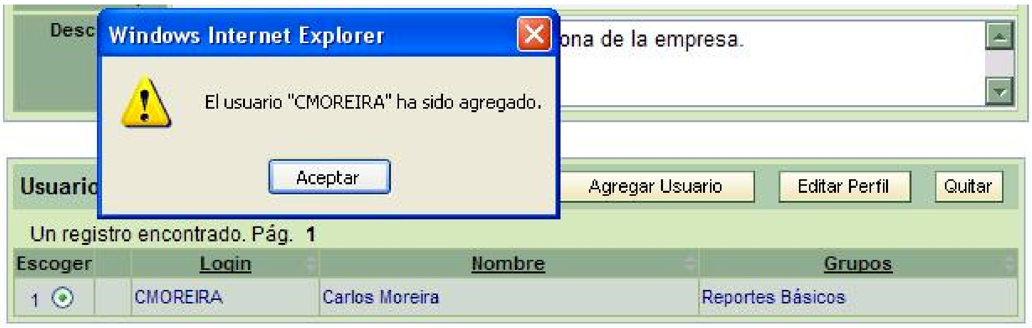


Figura 58: Prueba de incluir a un usuario en un grupo

Crear un reporte básico y diseñarlo con Eguana Reports

En la opción de Administrar Reportes creamos un nuevo reporte que se llame “Lista de Usuarios de Eguana Reports”. Se asume que la fuente de datos ya está creada.

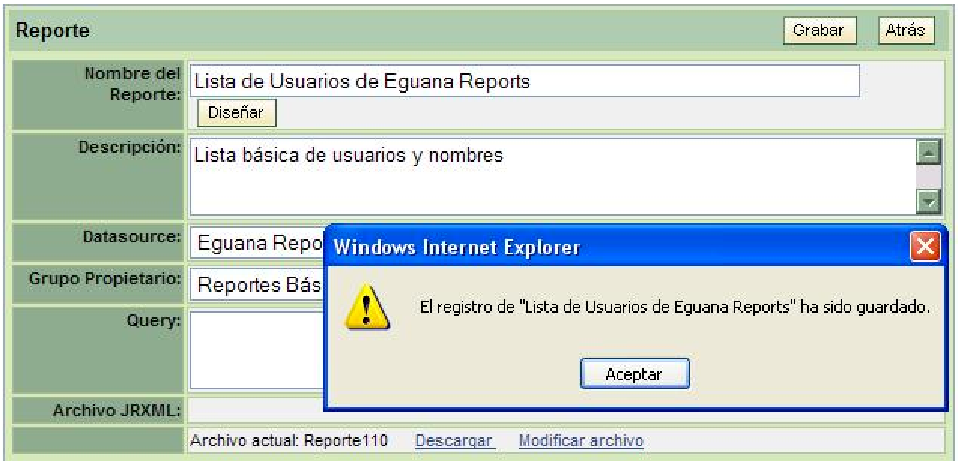


Figura 59: Prueba de creación de un reporte

En el diseño con Eguana Reports, se escogen los campos a mostrar.

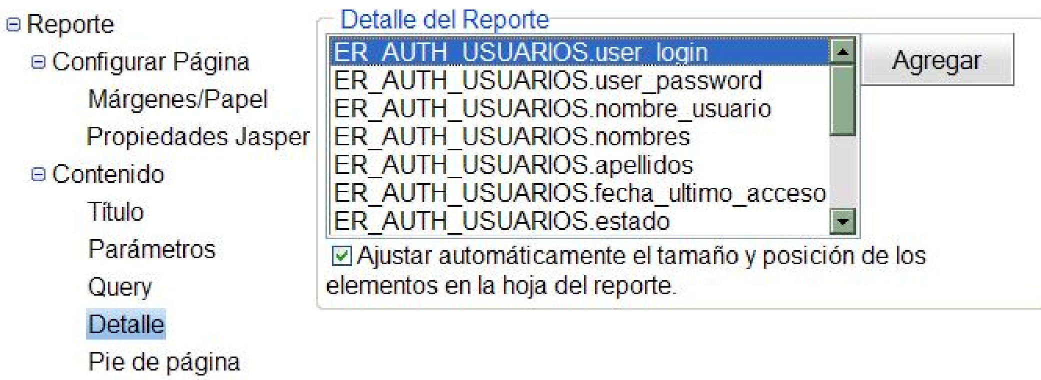


Figura 60: Prueba de diseño con Eguana Reports

En la ventana de mantenimiento del reporte descargar el archivo de diseño y modificarlo un poco para mejorar el estilo. Luego lo volvemos cargar utilizando la opción de “modificar archivo”.

Obtener reporte

Volvemos ingresar al sistema como con el usuario CMOREIRA, y escoger la opción “Obtener Reportes”. En la lista debe aparecer el reporte que ingresamos. Hacer clic en el nombre y escoger el formato de salida, a PDF.

Debemos obtener lo siguiente:

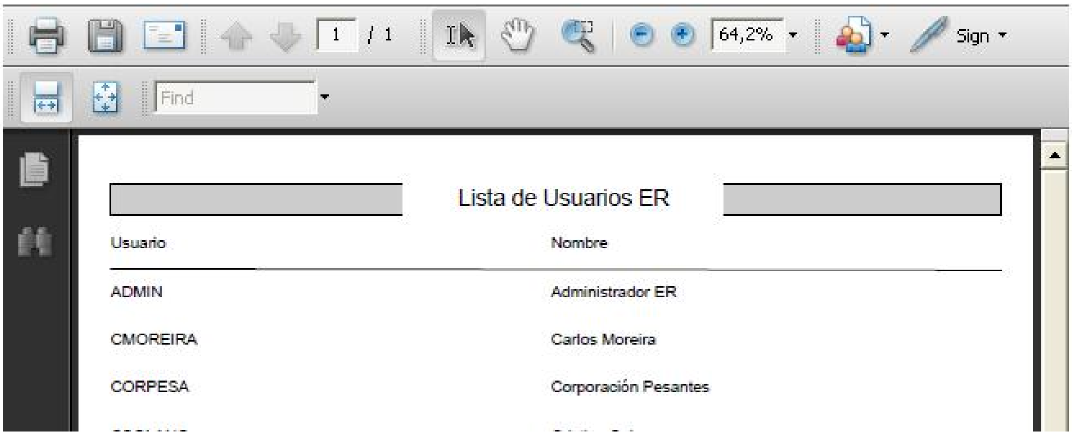


Figura 61: Reporte obtenido

CONCLUSIONES

Según nuestra experiencia en el desarrollo de Eguana Reports podemos concluir lo siguiente:

1. El desarrollo de aplicaciones en Java y código abierto tiene sus ventajas y desventajas. La disponibilidad de software para cumplir cualquier objetivo es alta, pero es eso mismo lo que dificulta la tarea de elegir qué herramienta usar. La tarea de selección muestra que, a diferencia de los demás módulos de Eguana, no fue necesario usar EJBs para implementar la solución a nuestro modelo. En vez de eso, utilizamos Hibernate.

   Otra herramienta que tuvimos que cambiar es SOFIA, que es gratuita, pero es una solución propuesta, en principio, por una empresa de desarrollo como solución a sus propios problemas. Ciertas tareas de implementación se nos hicieron imposibles con SOFIA y fue reemplazada por Struts para implementar MVC. Struts tiene mejor flexibilidad e integración con otras tecnologías y herramientas, además de mejor soporte para ayuda y aceptación dentro de los desarrolladores.

2. En el código abierto se puede obtener el código fuente, es verdad, pero alterarlo requiere de un desarrollador avanzado.

3. En el mundo de código abierto las actualizaciones son más rápidas que con código propietario. Esto se debe a la cantidad de gente colaborando, muchos gratis y otros como parte de las fundaciones de código abierto.

4. La rapidez hace que aparezcan nuevas herramientas día a día. Ya en la actualidad las versiones usadas en nuestro proyecto no son las últimas, e incluso existen otras tecnologías como JSF (Java Server Faces).

5. Aunque la curva de aprendizaje es pronunciada, el uso de Java EE y XML en la arquitectura de la aplicación asegura que sea escalable y portable, y que se ajuste a las necesidades de crecimiento futuro en una empresa.

6. Eguana Reports demuestra la versatilidad de JasperReports para generar reportes, que es una razón de que sea una herramienta muy utilizada hoy en día. Eguana Reports se basa en Jasper Reports para crear un ambiente más dinámico en la generación de un reporte, disminuyendo el tiempo que toma diseñar el mismo y obtener resultados.

7. Generar reportes en varios formatos permite que este proyecto se adapte a las necesidades del cliente al momento de elegir el programa donde desea ver los reportes, de una manera sencilla y dinámica.

8. Tener Eguana Reports, como servidor de reportes, permite a una empresa centralizar el almacenamiento y generación de reportes, optimizando los recursos, esfuerzos y organizando la tarea de reportería en un solo lugar.

9. Eguana Reports provee un entorno seguro de acceso a los reportes mediante usuarios, grupos y permisos. Dado que Eguana Reports centraliza el almacenamiento y generación de reportes, se puede implementar un mejor esquema de seguridad. Además se pueden distribuir los reportes por grupos o departamentos de una empresa y controlar mejor el acceso a reportes.

10. Con la creación del rol Administrador de Reportes en una empresa, es posible abstraer a los desarrolladores del problema de incluir reportes, o los mecanismos necesarios para generarlos, en cada módulo de la aplicación y así evitar la desorganización.

11. El prototipo de diseñador de reportes incluido en Eguana Reports permite crear reportes básicos aún más rápido que tan sólo utilizando Jasper Reports, sin necesidad de programar y con pocos conocimientos en diseñar reportes.

12. Hemos demostrado, a lo largo de este documento, la estructura independiente de Eguana Reports. Esta característica permite elevar la disponibilidad de reportes en una empresa.

RECOMENDACIONES

1. Con respecto al desarrollo con código abierto, Java EE y XML, se recomienda buscar la mayor cantidad de ayuda posible cuando no se está claro sobre las tecnologías involucradas, ya sea a profesores, profesionales y otros desarrolladores avanzados. Además, se debe leer muy bien la documentación y estar preparados para afrontar cualquier tipo de obstáculo y problemas con la utilización de herramientas de código abierto. Esta es una carrera de paciencia.

2. Sería recomendable que el desarrollador forme parte de un grupo (de desarrollo) que implemente soluciones en el mercado, en la vida real. La mejor forma de aprender es haciendo y probando que lo que uno hace sirve. No debe el desarrollador intentar implementar una solución compleja en la vida real sin tener una experiencia previa y los conocimientos y ayuda necesarios.

3. Con respecto a nuestra aplicación, se recomienda su mejora constante en la aceptación de diseños de versiones distintas de JasperReports. JasperReports se actualiza constantemente y los formatos de diseño cambian.

4. Se recomienda también la mejora del prototipo de diseñador. En este momento provee un diseño elemental. Aún así, la base para el crecimiento está estructurada desde el diseño gracias al uso de XML.

Por lo demás, el desafío de desarrollar con Java y XML, y en general de aplicaciones web, dentro del mundo de código abierto y colaborativo puede ser bastante estimulante. En esa tarea les deseamos éxito.

ANEXOS

ANEXO 1

DESCRIPCIÓN DE LAS TABLAS

ER_AUTH_USUARIOS

Usuario para ingreso al sistema.

Nombre del campo	Tipo / descripción
user_login	varchar(16) NOT NULL auto_increment Login del usuario de reportes.
user_password	varchar(16) NOT NULL default ''
nombre_usuario	varchar(100) NOT NULL default '' Nombre o sobrenombre del usuario.
nombres	varchar(100)
apellidos	varchar(100) Nombre completo, para efectos descriptivos.
fecha_ultimo_acceso	Datetime
estado	char(1) 'A': activo, 'I': inactivo

PRIMARY KEY (`user_login`)

ER_AUTH_ROLES

Roles del sistema.

Nombre del campo	Tipo / descripción
id_rol	int(9) NOT NULL auto_increment
nombre	varchar(100) NOT NULL default ''
descripcion	varchar(255)
orden	int(5)
estado	char(1)

	'A': activo, 'I': inactivo

PRIMARY KEY (`id_rol `)

ER_AUTH_OPCIONES

Opciones disponibles en el sistema.

Nombre del campo	**Tipo / descripción**
id_opcion	int(9) NOT NULL auto_increment
nombre_opcion	varchar(255) NOT NULL default ''
web_path	varchar(255) Directorio, dentro del contexto, donde se encuentran todos los archivos relacionados con esta opción que puede acceder. Ej.: /public/jsp*global/*
mostrable	char(1) NOT NULL Si esta opción se muestra en pantalla. Una opción que no se muestra sirve para otorgar permisos de acceso a un path.
url	varchar(255) Qué se ejecuta cuando el usuario escoge esta opción.
id_opcion_depende_de	numeric(9) Para agrupar opciones con permisos a diferentes recursos (diferentes web_path)
estado	char(1) 'A': activo, 'I': inactivo

PRIMARY KEY (`id_rol `)

ER_AUTH_ROLES_OPCIONES

Relación entre opciones y roles para indicar qué opciones tiene un rol.

Nombre del campo	**Tipo / descripción**
id_rol	numeric(9) NOT NULL

id_opcion	numeric(9) NOT NULL
PRIMARY KEY (\`id_rol\`, \`id_opcion\`)	

ER_AUTH_USUARIOS_ROLES

Relación entre opciones y roles para indicar qué opciones tiene un rol.

Nombre del campo	**Tipo / descripción**
user_login	varchar(16) NOT NULL
id_rol	numeric(9) NOT NULL
PRIMARY KEY (\`id_rol\`, \`user_login \`)	

ER_DATASOURCES

Una fuente de datos de la cual se obtiene información para generar uno o más reportes.

Nombre del campo	**Tipo / descripción**
id_datasource	int(11) NOT NULL auto_increment La clave primaria, identificador único.
nombre	varchar(255) NOT NULL default '' Nombre único de la fuente de datos.
descripción	
tipo_recurso	varchar(4) NOT NULL ‘JDBC’ o ‘JNDI’ Si es JNDI, utiliza el campo “nombre”. El resto de campos no se necesitan. Si es JDBC el resto de campos son necesarios.
driver	varchar(255) NOT NULL default ''
url	varchar(255) NOT NULL default ''
usuario	varchar(255) NOT NULL default ''
password	varchar(255) NOT NULL default ''

maxIdle	int(11) NOT NULL default '0'
maxActive	int(11) NOT NULL default '0'
maxWait	int(11) NOT NULL default '0'
query_validacion	varchar(255) NOT NULL default ''
id_usuario_ingreso	varchar(16) NOT NULL seguridad: usuario que ingresó el registro
id_usuario_imodificacion	varchar(16) seguridad: ultimo usuario que modificó el registro
fecha_ingreso	datetime NOT NULL seguridad: fecha de ingreso del registro
fecha_modificacion	datetime seguridad: última fecha de modificación del registro
estado	char(1) 'A': activo, 'I': inactivo
PRIMARY KEY (`id_datasource`)	

ER_USUARIOS

Usuario de reportes.

Nombre del campo	Tipo / descripción
user_login	varchar(16) NOT NULL auto_increment Login del usuario de reportes.
email	varchar(20) NOT NULL default ''
id_usuario_ingreso	varchar(16) NOT NULL seguridad: usuario que ingresó el registro
id_usuario_imodificacion	varchar(16) seguridad: ultimo usuario que modificó el registro
fecha_ingreso	datetime NOT NULL seguridad: fecha de ingreso del registro
fecha_modificacion	datetime seguridad: última fecha de modificación del registro
estado	char(1) 'A': activo, 'I': inactivo

```
PRIMARY KEY  (`user_login`)
```

ER_GRUPOS

Grupo de usuarios de reporte.

Nombre del campo	Tipo / descripción
id_grupo	int(9) NOT NULL auto_increment La clave primaria, identificador único.
nombre	varchar(255) NOT NULL default '' Nombre del grupo
descripcion	varchar(255)
id_usuario_ingreso	varchar(16) NOT NULL seguridad: usuario que ingresó el registro
id_usuario_imodificacion	varchar(16) seguridad: ultimo usuario que modificó el registro
fecha_ingreso	datetime NOT NULL seguridad: fecha de ingreso del registro
fecha_modificacion	datetime seguridad: última fecha de modificación del registro
estado	char(1) 'A': activo, 'I': inactivo

```
PRIMARY KEY  (`id_grupo`)
```

ER_USUARIOS_GRUPOS

Tabla de asociación entre usuario de reportes y grupo de reportes.
Un usuario puede estar en muchos grupos. Un grupo puede tener muchos usuarios.

Nombre del campo	Tipo / descripción
id_usuario_grupo	int(9) NOT NULL auto_increment La clave primaria, identificador único.
id_grupo	int(9) NOT NULL

user_login	varchar(16) NOT NULL auto_increment
administrador	varchar(1) NOT NULL Administrador de grupo. S: Sí; N: No
acceso	varchar(1) NOT NULL default ‘P’ Si el grupo tiene acceso al reporte. P:permitido; N:negado
id_usuario_ingreso	varchar(16) NOT NULL seguridad: usuario que ingresó el registro
id_usuario_imodificacion	varchar(16) seguridad: ultimo usuario que modificó el registro
fecha_ingreso	datetime NOT NULL seguridad: fecha de ingreso del registro
fecha_modificacion	datetime seguridad: última fecha de modificación del registro
estado	char(1) 'A': activo, 'I': inactivo
PRIMARY KEY (`id_usuario_grupo`)	

ER_REPORTES

Un reporte en Eguana Reports, que puede o no tener el query (a partir del cual se obtienen los datos) embebido. Los datos se obtienen conectándose a la fuente de datos a la que el reporte ha sido asociado.
Los reportes se asocian directamente a los grupos, no a los usuarios.
Pueden ser presentados en formato PDF, CSV, XLS y HTML.

Nombre del campo	Tipo / descripción
id_reporte	int(9) NOT NULL auto_increment
nombre	varchar(256) NOT NULL default '' Nombre único.
descripcion	varchar(255)
politica_defecto	varchar(1) NOT NULL default ‘P’ Política por default para un grupo que no sea dueño del reporte. P:permitido; N:negado
query	text (opcional) el query (para obtener los datos) que se envía al reporte compilado, en caso de que el query no esté embebido en el mismo reporte compilado. Es una opción de JasperReports.
query_xml	text representaciónen XML del query del reporte. No

	tiene relación con query. Es para diseñar el query con el GUI.
id_archivo	numeric(9) NOT NULL Referencia a archivo.
id_datasource	int(9) NOT NULL default '0' Referencia al datasource del cual se obtienen los datos para llenar el reporte
id_usuario_ingreso	varchar(16) NOT NULL seguridad: usuario que ingresó el registro
id_usuario_imodificacion	varchar(16) seguridad: ultimo usuario que modificó el registro
fecha_ingreso	datetime NOT NULL seguridad: fecha de ingreso del registro
fecha_modificacion	datetime seguridad: última fecha de modificación del registro
estado	char(1) 'A': activo, 'I': inactivo
PRIMARY KEY (`id_reporte`)	

ER_REPORTES_GRUPOS

Tabla de asociación entre el reporte y grupo de usuarios.
¿Qué reportes están asignados a un grupo X?
¿Qué grupos pueden obtener un reporte A?

Nombre del campo	Tipo / descripción
id_reporte_grupo	int(9) NOT NULL
id_grupo	int(9) NOT NULL
id_reporte	int(9) NOT NULL
propietario	varchar(1) NOT NULL default ‘1’ Si el grupo es propietario del reporte.
acceso	varchar(1) NOT NULL default ‘P’ Si el grupo tiene acceso al reporte. P:permitido; N:negado
id_usuario_ingreso	varchar(16) NOT NULL seguridad: usuario que ingresó el registro
id_usuario_imodificacion	varchar(16) seguridad: ultimo usuario que modificó el registro
fecha_ingreso	datetime NOT NULL seguridad: fecha de ingreso del registro

fecha_modificacion	datetime seguridad: última fecha de modificación del registro
estado	char(1) 'A': activo, 'I': inactivo
PRIMARY KEY (`id_reporte_grupo`)	

ER_PARAMETROS

Un reporte puede (o no) tener parámetros. Esta tabla guarda todos los parámetros cuyo valor debe ser ingresado cuando se solicite un reporte.

Nombre del campo	**Tipo / descripción**
id_parametro	int(9) NOT NULL
id_reporte	int(9) Nunca es null, pero se permite por problemas con Hibernate. Hibernate se encarga de la integridad.
nombre	varchar(100) NOT NULL default '' Nombre del parámetro y nombre del componente HTML que representa el valor a ingresar.
nombre_variable	varchar(100) NOT NULL Nombre de la variable de reporte tal como la recibe el archivo JRXML.
descripcion	varchar(255)
label	varchar(100) NOT NULL default 'Ingrese' La etiqueta que va justo antes del input HTML
requerido	varchar(1) NOT NULL 1: Es requerido(obligatorio); 0: No es requerido
tipo_elemento_HTML	varchar(1) NOT NULL Tipo de elemento HTML. I: input text; L: lista opción única; M: lista opción múltiple; S: select combo; R: radio; C: checkbox
classname	varchar(3) NOT NULL default '' Tipo de dato Java del parámetro que debe pasarse al reporte. 'BIG': BigDecimal, 'DOU': Double, 'FLO': Float, 'LON': Long, 'INT': Integer, 'SHO': Short, 'BOO': Boolean
orden	int(5)

	Posición con respecto a los demás parámetros del mismo reporte en el layout para petición de parámetros.
id_usuario_ingreso	varchar(16) NOT NULL seguridad: usuario que ingresó el registro
id_usuario_imodificacion	varchar(16) seguridad: ultimo usuario que modificó el registro
fecha_ingreso	datetime NOT NULL seguridad: fecha de ingreso del registro
fecha_modificacion	datetime seguridad: última fecha de modificación del registro
estado	char(1) 'A': activo, 'I': inactivo

PRIMARY KEY (`id_parametro`)

ER_VALORES_PARAMETROS

Un parámetro es llenado vía web y enviado para obtener el reporte.
Los componentes web para obtener un parámetro pueden ser: text, select, combo o radiogroup.
En el caso de TEXT value representa el valor por defecto que se muestra al momento de pedir el ingreso.
En el caso de los demás, al ser como listas (combo, select, radiogroup), los valores a seleccionar poseen un identificador que programáticamente sirve para determinar el valor escogido (value), mientras al usuario se le muestra un texto amigable (textoAPresentar)

Nombre del campo	**Tipo / descripción**
id_valor_parametro	int(9) NOT NULL
id_parametro	int(9) NOT NULL El parámetro al que pertenece el valor.
tipo	varchar(1) E: Estático, D: Dinámico (se sacan los valores con un query)
label	varchar(255) Lo que se presenta visualmente (ej.: en un <select>) Texto que se presenta al usuario.
valor	varchar(255) El atributo value en el elemento HTML

selected	varchar(1) Indica si el valor es el seleccionado por defecto dentro de una lista, combo o radiogroup.
query	text Query para valores dinámicos.
order_by	varchar(255) Aplicable al query de los valores de parámetro dinámicos. Indica los campos de ordenamiento para el query.
orden	int(5) Posición con respecto a los demás parámetros del mismo reporte en el layout para petición de parámetros.
id_usuario_ingreso	varchar(16) NOT NULL seguridad: usuario que ingresó el registro
id_usuario_imodificacion	varchar(16) seguridad: ultimo usuario que modificó el registro
fecha_ingreso	datetime NOT NULL seguridad: fecha de ingreso del registro
fecha_modificacion	datetime seguridad: última fecha de modificación del registro
estado	char(1) 'A': activo, 'I': inactivo

PRIMARY KEY (`id_valor_parametro`)

ER_ARCHIVOS

Referencia a un archivo en disco.

Nombre del campo	Tipo / descripción
id_archivo	numeric(9) NOT NULL
nombre	varchar(256) NOT NULL default '' Nombre único.
descripcion	varchar(255)
url	varchar(255) Ubicación del archivo. FULL PATH.
extension_archivo	varchar(5) NOT NULL
autor	varchar(100)
content_type	varchar(50)

id_usuario_ingreso	varchar(16) NOT NULL seguridad: usuario que ingresó el registro
id_usuario_imodificacion	varchar(16) seguridad: ultimo usuario que modificó el registro
fecha_ingreso	datetime NOT NULL seguridad: fecha de ingreso del registro
fecha_modificacion	datetime seguridad: última fecha de modificación del registro
estado	char(1) 'A': activo, 'I': inactivo

PRIMARY KEY (`id_archivo`)

ER_CLAVES_PRIMARIAS

Tabla donde se guarda la secuencia de claves primarias para las tablas. Es de uso específico de Hibernate con un generador de claves diseñado para el caso.

Nombre del campo	**Tipo / descripción**
id_archivo	numeric(9)
id_rol	numeric(9)
id_grupo	numeric(9)
id_datasource	numeric(9)
id_reporte	numeric(9)
id_parametro	numeric(9)
id_valor_parametro	numeric(9)
id_usuario_grupo	numeric(9)
id_reporte_grupo	numeric(9)

ANEXO 2

HIBERNATE EN DETALLE

¿Qué es Hibernate?
Es una herramienta de mapeo objeto/relacional para ambiente Java.
El término mapeo objeto/relacional (ORM, por sus siglas en inglés) se refiere a la técnica para la representación de datos de un modelo de objetos a un modelo de datos relacional con un esquema basado en SQL.

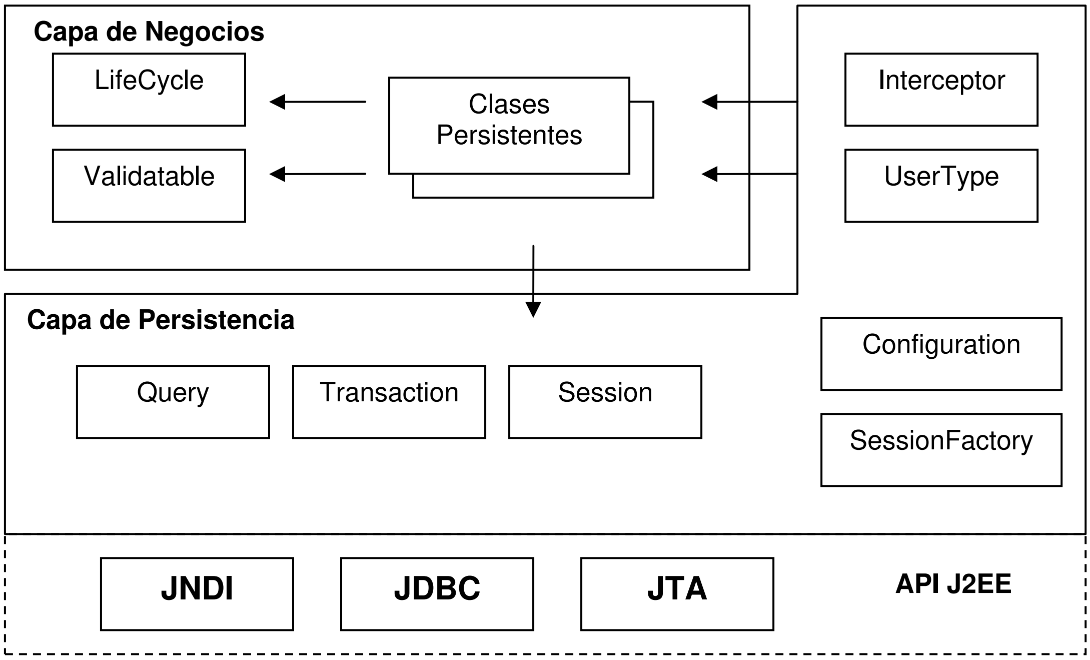
¿Por qué se usa Hibernate?
Hibernate propone una forma de trabajar con objetos y hacer la persistencia en una base de datos relacional.
El objetivo es liberar al desarrollador de la mayoría de las tareas comunes de persistencia en la programación. Además que es una alternativa más simple al uso de EJBs.

Sin embargo, hay que considerar al momento de diseño que Hibernate (capa de acceso a datos) estará fuertemente integrado a la aplicación.

Tomcat con Hibernate
Tomcat no es un servidor de aplicaciones completo. Es sólo un contenedor de servlets, aunque con algunas características que se encuentran en los servidores de aplicaciones.
Una de estas características es el pool de conexiones. Tomcat utiliza DBCP, y se expone a las aplicaciones como un recurso JNDI, tal como lo hacen los servidores de aplicaciones.
Sin embargo Tomcat no provee de un administrador de transacciones. Implica que hay que proporcionarle las librerías necesarias del API de JTA para que Hibernate pueda manejar las transacciones adecuadamente.

ARQUITECTURA

Para utilizar Hibernate en la capa de persistencia hay que entender su arquitectura, aunque sea de una manera básica.



- Las interfaces que utilizan las aplicaciones para hacer las operaciones comunes y queries son Session, Transaction, y Query.
- Entre las interfaces que se utilizan para configurar Hibernate están Configuration y SessionFactory.
- Las interfaces llamadas de *callback* (LifeCycle, Interceptor, y demás) permiten a la aplicación reaccionar ante eventos generados dentro de Hibernate.
- Las interfaces que sirven para extender la funcionalidad de Hibernate incluyen UserType, CompositeUserType, IdentifierGenerator.

Interfaces

INTERFACES PRINCIPALES
Session
Puede pensarse en una sesión como un caché de objetos persistentes relacionados con una unidad de trabajo. En él Hibernate detecta cambios en los objetos, y realiza operaciones de persistencia de objetos en la base.

Puede también pensarse en una sesión como el *administrador de persistencia*.
Una instancia de Session es ligera y no es costosa para crear o destruir. Esto es importante ya que la aplicación necesita crear y destruir sesiones a cada momento. No son *threadsafe* y deberían ser usadas sólo por un thread (hilo).

SessionFactory
La aplicación obtiene una instancia de Session de un SessionFactory.
Una instancia de SessionFactory está diseñada para ser utilizada por varios hilos. Lo mejor es utilizar una sola instancia en toda la aplicación por cada base de datos a accesar.
Un SessionFactory mantiene las sentencias SQL generadas y otra metadata que Hibernate utiliza al momento de ejecución. También puede mantener datos ya leídos en una unidad de trabajo para que estén disponibles para otra unidad de trabajo, con una configuración adecuada (second-cache level).

Configuration
La aplicación utiliza una instancia de Configuration para localizar los archivos de configuración y mapeo para luego crear una instancia de SessionFactory.

Transaction
Hibernate da la opción para obviar el manejo de transacciones mediante el código utilizando la interfaz Transaction. Se puede preferir, sin embargo, manejar las transacciones mediante código específico en la aplicación.
Este API para transacciones hace que el código sea portable entre varios contenedores y ambientes de ejecución.

Query
Permite hacer consultas a la base utilizando lenguaje HQL o, si se lo desea, en el código SQL nativo de la base de datos.
Una instancia de Query no puede ser utilizada fuera de la instancia Session donde fue creada.

INTERFACES CALLBACK
La propiedad de “callback” permite a la aplicación recibir notificaciones cuando algún evento específico ocurre dentro de Hibernate. Un evento puede ser la creación, modificación o eliminación de un objeto.
Las interfaces Lifecycle y Validatable permiten a un objeto persistente reaccionar a eventos relacionados con su propio ciclo de vida. Sin embargo, este código haría que la clase no sea portable fuera de Hibernate.
Interceptor es una interfaz que soluciona el problema de portabilidad. Permite el callback sin necesidad de que las clases adopten una interfaz específica del API de Hibernate.

TYPES
Un objeto Type mapea un tipo de dato Java a un tipo de dato de la base de datos, incluyendo sus asociaciones.
También soporta tipos de datos definidos por el usuario, mediante las interfaces `UserType` y `CompositeUserType`.

INTERFACES DE EXTENSION
Junto con la funcionalidad principal, el mapeo, existen otras extensiones de funcionalidad para reforzar las características de Hibernate.
Mucha de la funcionalidad de Hibernate es configurable, y muchas veces es necesario proveer puntos de extensión a la funcionalidad con el uso de interfaces:

- Generador de clave primaria (interfaz `IdentifierGenerator`).
- Dialecto SQL (clase abstracta `Dialect`).
- Caché.
- Manejo de conexiones JDBC (interfaz `ConnectionProvider`).
- Manejo de transacciones (`Transaction`, `TransactionFactory`, `TransactionFactoryLookup`).
- Estrategias de mapeo objeto-relacional (`ClassPersister`).
- Acceso a Propiedades (`PropertyAccesor`).
- Creación de proxy (`ProxyFactory`)

CONFIGURACIÓN

Para hacer funcionar Hibernate hay que saber configurarlo. Hibernate puede funcionar en casi todo ambiente Java. Usualmente se utiliza en aplicaciones cliente/servidor de dos y tres capas.

- Managed environment: Define límites de transacciones, permite declarar la seguridad y mantiene recursos (en pools) , típicamente como conexiones a la base de datos. Ejemplos de un ambiente de este tipo son JBOSS, WebSphere y WebLogic.
- Non-managed environment: Provee manejo básico de concurrencia a través de pool de threads. Jetty, Tomcat, aplicaciones stand-alone son ejemplos de este tipo de ambiente. Estos ambientes no proveen infraestructura de seguridad o manejo automático de recursos y transacciones. La aplicación en sí debe marcar estos límites y manejar las transacciones y conexiones.

En este punto, Hibernate trata de abstraer el ambiente en el cual la aplicación está siendo puesta en producción. En el caso de Tomcat, por ejemplo, Hibernate se encarga de manejar las transacciones y las conexiones JDBC. En el caso de JBoss, Hibernate se integra con el ambiente que éste le provee para el manejo de conexiones y transacciones.

Sea cual sea el caso, lo primero para utilizar Hibernate es crear una instancia de `SessionFactory` a partir de `Configuration`.

Crear SessionFactory

Una instancia de `net.sf.hibernate.cfg.Configuration` representa básicamente un conjunto de mapeos de tipos de datos Java a tipos de dato SQL.
A partir de una instancia de `Configuration` se puede obtener una instancia de `SesisonFactory`. La configuración debe levantarse una sola vez por aplicación y por base de datos, y se encarga de ubicar los archivos de mapeo.

```
Configuration config = new Configuration();
config.setProperties( System.getProperties() );
SessionFactory sessionFactory =
config.buildSessionFactory();
```

Los archivos de mapeo de Hibernate tienen la extensión hbm.xml por defecto. La práctica recomendada es tener un archivo de mapeo por cada clase persistente, que debe estar ubicado en el mismo directorio que la clase (`.class`).

La configuración abarca muchas opciones, las cuales en su mayor parte se establecen con valores por defecto adecuados. Para establecer propiedades se pueden utilizar dos métodos, considerados los más adecuados:

- Poner un archivo llamado hibernate.properties en el classpath.
- Poner tags `<property>` dentro de `hibernate.cfg.xml` en el classpath.

Ambas proveen la función de configurar Hibernate.
El Javadoc para la clase `net.sf.hibernate.cfg.Environment` documenta cada propiedad de configuración de Hibernate.

La configuración más usual es la conexión a la base de datos, y tal vez la más importante.
Hay que tener en cuenta si el ambiente será administrado (manager environment) o no-administrado (non-managed environment).

Ambiente no-administrado
En un ambiente no administrado la aplicación debe preocuparse por implementar su esquema de conexión a la base y el pool de conexiones.
Los "pools" de conexiones soportados por Hibernate son Apache DBCP, 3CP0 y Proxool. La comunidad de Hibernate tiende a preferir 3CP0 o Proxool.

Los pasos básicos para configurar Hibernate se resumen:

- Colocar tanto `hibernate2.jar` como el driver para conexión a la base en el classpath de la aplicación.
- Colocar las librerías de las que depende Hibernate en el directorio /lib de la aplicación.
- Escoger un pool de conexiones JDBC soportado por Hibernate y configurarlo adecuadamente con un archivo de propiedades.
- Escribir las propiedades de la conexión y demás propiedades en el archivo `hibernate.properties` o `hibernate.cfg.xml` en el classpath.
- Crear una instancia de Configuration, que cargará los archivos de mapeo. Luego crear una instancia de SessionFactory a partir de la configuración utilizando el método `buildSessionFactory()`.

Ambiente administrado
Un ambiente administrado típicamente maneja ciertos aspectos avanzados de la aplicación como seguridades, manejo de transacciones y administración de recursos (pool de conexiones).
Aunque generalmente están diseñados para soportar EJBs, se puede tomar ventaja de estas características aunque no se utilicen EJBs.
El API de Hibernate que se utiliza es el mismo que con servlets o JSPs.

Un servidor de aplicaciones habilita un pool de conexiones JDBC como un recurso JNDI. Este recurso es una instancia de `javax.jdbc.Datasource`. Debe indicársele a Hibernate, en el archivo de propiedades (`hibernate.properties` o `hibernate.cfg.xml`), el nombre JNDI bajo el cual localizar el recurso para acceder a la base de datos.

En Java ya existe el estándar para manejo de transacciones, JTA, que se utiliza para controlar las transacciones en los ambientes administrados. Esto se llama *container-managed transactions* (CMT). Si JTA está presente, las conexiones JDBC pasan bajo el total control de este administrador.
Es evidente que ambientes administrados y no-administrados pueden utilizar esquemas de transacciones distintos. Entonces Hibernate provee un API para controlar transacciones, con el fin de ser protable entre ambientes. Este API abstrae el tipo de transacción que hay por debajo y se la define mediante

la propiedad `hibernate.connection.factory_class`, que puede tomar uno de los dos valores siguientes:

- `net.sf.hibernate.transaction.JDBCTransactionFactory` para delegar transacciones directamente por JDBC. Esta estrategia es para ambientes no-administrados y es el mecanismo por default, a no ser que se indique lo contrario.
- `net.sf.hibernate.transaction.JTATransactionFactory` delega las transacciones a JTA. Es la estrategia para ambientes administrados (CMT)

La configuración en hibernate.properties sería:

```
hibernate.connection.datasource =
java:/comp/env/jdbc/Eguana
hibernate.transaction.factory_class =
net.sf.hibernate.transaction.JTATransactionFactory
hibernate.transaction.manager_lookup_class =
net.sf.hibernate.transaction.JBossTransactionManager
Lookup
hibernate.dialect =
net.sf.hibernate.dialect.MySQLDialect
```

O también se puede usar, como ya se dijo, el archivo hibernate.cfg.xml, poniendo las propiedades en el correcto formato xml.

Siempre debe establecerse el dialecto SQL a una subclase de `net.sf.hibernate.dialect.Dialect`. A partir de éste Hibernate puede dar valores específicos de algunas de las propiedades de la configuración de Hibernate.

CONFIGURACION AVANZADA

Una vez que se tiene creado un pequeño ejemplo con Hibernate, es conveniente darle un vistazo a las demás propiedades de configuración de Hibernate (ver Javadoc para `net.sf.hibernate.cfg.Environment`).
Hay un parámetro importante en este punto, que se necesitará eventualmente en las aplicaciones desarrolladas con Hibernate. La propiedad `hibernate.show_sql` permite, cuando se establece en true, registrar todos los comandos SQL generados a la consola. Esto es importante para rastreo de errores y monitoreo.

Utilizar configuración con un archivo XML

La configuración se puede hacer programáticamente modificando propiedades de la instancia de Configuration, o estableciendo parámetros en un archivo `hibernate.properties`, o utilizando la configuración xml en el archivo `hibernate.cfg.xml`.
La mayoría de las veces se considera mejor configurar Hibernate con xml. Además que en este archivo se pueden especificar las rutas a los archivos de mapeo.
Ej.:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration
PUBLIC "-//Hibernate/Hibernate Configuration
DTD//EN"
"http://hibernate.sourceforge.net/hibernate-
configuration-2.0.dtd">

<hibernate-configuration>
     <!-- A SessionFactory is Hibernate's concept of
a single datastore -->
     <session-factory>
          <!-- Utilizar el DBCP declarado en
conf/server.xml -->
          <property
name="connection.datasource">java:comp/env/jdbc/Egua
na</property>
          <property name="show_sql">false</property>
          <property
name="dialect">net.sf.hibernate.dialect.MySQLDialect
</property>
          <!-- Mapping files -->
          <mapping resource="mapeo/Banco.hbm.xml"/>
     </session-factory>
</hibernate-configuration>
```

- La declaración de tipo de documento es necesaria para que el parser XML valide este documento contra el DTD de Hibernate.
- Se declara una fábrica de sesiones. Opcionalmente se puede dar un nombre (atributo `name`) para usarlo con JNDI.
- Las propiedades siguen a continuación. Son similares a las utilizadas en el archivo hibernate.properties.

- Se especifican los archivos de mapeo para las clases persistentes. La ruta es relativa al archivo de configuración.

El archivo xml debe cumplir con el dtd (`hibernate-configuration-2.0.dtd`):

```
<!ELEMENT hibernate-configuration (session-factory)>

<!ELEMENT property (#PCDATA)>
<!ATTLIST property name CDATA #REQUIRED>

<!ELEMENT mapping EMPTY> <!-- reference to a mapping
file -->
<!ATTLIST mapping resource CDATA #IMPLIED>
<!ATTLIST mapping file CDATA #IMPLIED>
<!ATTLIST mapping jar CDATA #IMPLIED>

<!ELEMENT jcs-class-cache EMPTY> <!-- deprecated -->
<!ATTLIST jcs-class-cache class CDATA #REQUIRED>
<!ATTLIST jcs-class-cache region CDATA #IMPLIED>
<!ATTLIST jcs-class-cache usage (read-only|read-
write|nonstrict-read-write) #REQUIRED>

<!ELEMENT jcs-collection-cache EMPTY> <!--
deprecated -->
<!ATTLIST jcs-collection-cache collection CDATA
#REQUIRED>
<!ATTLIST jcs-collection-cache region CDATA
#IMPLIED>
<!ATTLIST jcs-collection-cache usage (read-
only|read-write|nonstrict-read-write|transactional)
#REQUIRED>

<!ELEMENT class-cache EMPTY>
<!ATTLIST class-cache class CDATA #REQUIRED>
<!ATTLIST class-cache region CDATA #IMPLIED>
<!ATTLIST class-cache usage (read-only|read-
write|nonstrict-read-write|transactional) #REQUIRED>

<!ELEMENT collection-cache EMPTY>
<!ATTLIST collection-cache collection CDATA
#REQUIRED>
<!ATTLIST collection-cache region CDATA #IMPLIED>
<!ATTLIST collection-cache usage (read-only|read-
write|nonstrict-read-write|transactional) #REQUIRED>
```

```
<!ELEMENT session-factory (property*, mapping+,
(class-cache|collection-cache|jcs-class-cache|jcs-
collection-cache)*)>
<!ATTLIST session-factory name CDATA #IMPLIED> <!--
the JNDI name -->
```

En particular se debe tener en cuenta, según el dtd, que el archivo de configuración puede tener 0 o más elementos `property`, y debe tener al menos 1 elemento `mapping` (o sea referencia a un archivo de mapeo de clase persistente).

La inicialización de Hibernate es similar:

```
SessionFactory sessionFactory = new
Configuration().configure().buildSessionFactory();
```

Hibernate busca por defecto el archivo de `hibernate.cfg.xml` en el *classpath* de la aplicación. Si Hibernate encuentra el archivo `hibernate.properties` en el classpath, entonces reescribirá el valor de las propiedades con los especificados en el archivo de configuración xml.

La configuración vía xml permite es centralizada y evita el tener que configurar los archivos de mapeo en el código fuente.

Configuración de SessionFactory con JNDI

Para la mayoría de las aplicaciones sólo se necesita una instancia de SessionFactory.
JNDI es una forma de almacenar recursos y accesarlos a partir de una estructura jerárquica, ya sean objetos, referencias o atributos de configuración.
SessionFactory se enlazará al nombre JNDI si éste se especifica como atributo en la configuración xml:

```
<session-factory
name"java:/HibernateSessionFactory">
```

Tomcat viene con un contexto JNDI de sólo lectura. Hibernate no puede trabajar con este contexto, así que se debe utilizar un contexto completo (librería externa) o simplemente deshabilitar el uso de JNDI para la fábrica de sesiones.

Logging

Una aplicación que utiliza Hibernate generalmente no se preocupa por las sentencias que se ejecutan para llevar a cabo la persistencia.
Hibernate puede ser visto como una caja negra.
Las sentencias SQL son ejecutadas de una manera asíncrona, al final de las transacciones. Este es un comportamiento que se ha denominado, en inglés, *write-behind*.

A pesar de todo, a veces es necesario saber lo que ocurre detrás de escena cuando tratamos de resolver un problema difícil. Este es el propósito de registrar (log) las sentencias que se ejecutan.
El primer paso para hacer logging es establecer en true la propiedad `show_sql`. Esto permite registrar las sentencias SQL que Hibernate ejecuta. Por supuesto, esto no es suficiente para todos los casos.
Hibernate utiliza Apache *commons-logging*, una capa que abstrae a la aplicación de las diferentes herramientas para hacer logging. *Commons-loggin* actúa de intermediario para utilizar la herramienta de logging que mejor convenga. Por defecto utiliza *log4j* si lo encuentra en el classpath.

Para configurar log4j se necesita un archivo log4j.properties en el classpath, al mismo nivel que hibernate.cfg.xml. Una configuración simple de log4j sería:

```
### to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppend
er
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.Patter
nLayout
log4j.appender.stdout.layout.ConversionPattern=%d{AB
SOLUTE} %5p %c{1} - %m%n
log4j.rootLogger=error, stdout

log4j.logger.net.sf.hibernate=info
```

Reemplazando info por debug se obtiene más información.

Hibernate hace énfasis en hacer que los mensajes sean lo más detallados posible sin dejar que sean entendibles.

CLASES PERSISTENTES

Las clases persistentes representan las entidades del negocio. Típicamente se refieren a sustantivos (cuenta, banco, usuario) que se almacenan como un registro en una tabla.
Hibernate trabaja mejor con estas clases si se siguen unas reglas simples:

Declarar métodos de acceso y modificación a los atributos persistentes
Es decir, los métodos `get` y `set`.
Algunas herramientas ORM persisten directamente los atributos en la tabla. Sin embargo Hibernate recomienda separar este tipo de implementación del mecanismo de persistencia.

Implementar un constructor para la clase
Para que Hibernate pueda instanciar la clase. El constructor no necesariamente debe ser público.

Proveer un identificador
Esta propiedad equivale a la clave primaria de la tabla. Puede tener cualquier nombre y cualquier tipo de dato primitivo.

No utilizar clases final
La funcionalidad de proxies de Hibernate requiere que la clase no sea de tipo final, o que se implemente una interfaz cuyos métodos sean públicos.

Estas reglas simples corresponden a lo que se llaman modelo de programación POJO (Plain Old Java Object). Un ejemplo de esta clase es:

```
public class Grupo {
     private int id;
     private String nombre; // nombre completo del
grupo.
     private String descripcion;

     public Grupo(){
     }

     /**
      * Constructor.
      * @param nombre - nombre del grupo
      * @param descripcion
      */
```

```
	public Grupo(String nombre, String
descripcion){
		this.nombre = nombre;
		this.descripcion = descripcion;
	}
	public void setId(int id){
		this.id = id;
	}
	public void setNombre(String nombre){
		this.nombre = nombre;
	}
	public void setDescripcion(String descripcion){
		this.descripcion = descripcion;
	}
	public int getId(){
		return id;
	}
	public String getNombre(){
		return nombre;
	}
	public String getDescripcion(){
		return descripcion;
	}
}
```

MAPEO

El mapeo objeto/relacional para las clases persistentes se define en archivos xml. Son archivos diseñados para ser fácilmente manipulables y entendibles. El lenguaje de mapeo se centra en Java, es decir que describe el mapeo basándose en el punto de vista de cómo se persisten las clases en las tablas, y no en cómo los datos de las tablas se convierten en objetos.

Este es el archivo de mapeo para la clase persistente anterior:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-
//Hibernate/Hibernate Mapping DTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-
2.0.dtd">

<hibernate-mapping>
	<class name="com.ers.model.hibernate.Grupo"
table="grupo">
```

```
          <id name="id" type="int" unsaved-
value="null" >
               <column name="grupoId" sql-
type="int(11)" not-null="true"/>
               <generator class="increment"/>
          </id>
          <property name="nombre">
               <column name="nombre" length="100"/>
          </property>
          <property name="descripcion">
               <column name="descripcion"
length="255"/>
          </property>
     </class>
</hibernate-mapping>
```

Los elementos posibles dentro del mapeo se los describe en el DTD. Gran parte de ellos son fáciles de entender.

ID

Toda clase persistente debe declarar una columna de clave primaria dentro de la tabla de la base. En la mayoría de los casos la clase define un identificador para la instancia. El elemento `<id>` define el identificador de la clase mapeado a la tabla.

Los valores permitidos para unsaved-value son:
- `any`: siempre guarda el objeto
- `none`: siempre actualiza el objeto
- `null`: (predeterminado) guarda el objeto cuando el identificador es null.
- algún valor válido: guarda el objeto cuando el identificador es null o el valor dado.
- `undefined`: utiliza la `version o timestamp`, para luego chequear el identificador y grabar.

Un elemento importante es el generador, que es una clase que genera identificadores únicos para las instancias de las clases persistentes. Hay varios tipos de generadores, pero todos implementan la interfaz `net.sf.hibernate.id.IdentifierGenerator`. Los generadores están dentro del paquete `net.sf.hibernate.id`.

Generadores:

```
increment
identity
secuence
hilo
seqhilo
uuid.hex
uuid.string
native
assigned
foreign
```

En todos los casos Hibernate se encarga de generar el identificador (recomendado). Si no se desea esto, sin embargo, se lo puede asignar manualmente, especificando el generador `assigned`.

```
<generator class="assigned"/>
```

Si la clave es compuesta, entonces se utiliza el elemento `<composite-id>`:

```
<composite-id>
<key-property name="medicareNumber"/>
<key-property name="dependent"/>
</composite-id>
```

PROPIEDAD

Describe una propiedad(atributo) de la clase persistente al estilo de un JavaBean.

El tag `<property>` se utiliza:

```
<property
name="propertyName"
column="column_name"
type="typename"
update="true|false"
insert="true|false"
formula="arbitrary SQL expression"
access="field|property|ClassName"
/>
```

Los atributos usualmente utilizados son:

- `name` : nombre del atributo de la clase persistente.
- `column` : columna representativa en la tabla
- `type` : un tipo de dato Hibernate.

El tipo (`type`) puede ser:

- El nombre de un tipo de dato básico de Hibernate (`integer`, `string`, `character`, `date`, `timestamp`, `float`, `binary`, `serializable`, `object`, `blob`).
- El nombre de una clase Java con tipo básico de dato (`int`, `float`, `char`, `java.lang.String`, `java.util.Date`, `java.lang.Integer`, `java.sql.Clob`).
- El nombre de una subclase de `PersistentEnum` (ej.: `Color`).
- El nombre de una clase serializable Java.
- El nombre de un tipo definido por el usuario.

Si no se especifica un tipo Hibernate tratará de adivinar el tipo de dato utilizando “Java Reflection”.

ANEXO 3

SOFIA

(Salmon Open Framework for Internet Applications)

Sofia es un framework y librería de tags basado en J2EE diseñado para mejorar la productividad en el desarrollo de aplicaciones web, integrando lo mejor de las herramientas de desarrollo disponibles.
La filosofía es el desarrollo rápido de aplicaciones web (RAD- Rapid Application Development), basándose en ciertos principios:

- Utilizar la herramienta correcta para el trabajo correcto. Es decir, escribir HTML con un editor de HTML y código java con un editor de código java.
- Hacer las cosas más fáciles.
- Hacer que el desarrollador acepte los estándares de diseño donde sea necesario. Hacer que adoptar estos estándares no sea difícil. Darle también la opción al desarrollador para escoger la herramienta que mejor se ajuste a sus necesidades.
- Evitar que el desarrollador escriba código difícil y repetitivo.
- Evitar que el desarrollador se haga dependiente de un solo software. Conjugar las mejores herramientas siempre que sea posible.
- Hacer una herramienta flexible a las necesidades del desarrollador.
- Hacer que sea una herramienta fácil de aprender.
- Hacer lo posible por que la herramienta sea portable.

DISEÑO

Sofia hace énfasis en adoptar estándares de diseño en la arquitectura de la aplicación.

MVC (Model-View-Controller)

El modelo con el cual se separa la funcionalidad en una aplicación. Facilita el entendimiento en todo sentido y fomenta la reusabilidad de componentes.

El modelo corresponde a la parte de la aplicación donde se almacenan y manipulan datos (Ej.: EJBs, JavaBeans).
Vista corresponde a la parte con la que el usuario interactúa (Ej.: JSPs).
El controlador es quien se encarga de recoger las peticiones del usuario y realiza los cambios respectivos en el modelo. También se encarga de controlar el flujo de "vistas".

Sofia se basa en el modelo MVC y provee los componentes necesarios para desarrollar aplicaciones con esta arquitectura.

Componentes GUI inteligentes

Sofia provee de una librería de tags para crear componentes web y su respectiva representación en código HTML.

Patrón Observador

Para controlar los eventos Sofia utiliza el patrón llamado "Observador". Este patrón imita el comportamiento de Java AWT y Swing. Si algún componente necesita ser notificado de alguna acción de usuario, tal como hacer clic, llenar una forma, cambiar valores de algún campo, o cualquier otro, el componente será notificado cuando la acción ocurra.
Cualquier objeto que implemente apropiadamente la interface "event listener" puede escuchar cualquier evento que se genere en una página jsp.

Archivos de propiedades

Necesarios para mostrar toda la aplicación con un mismo estilo de interfaz de usuario. Usualmente una aplicación empresarial es desarrollada por muchas personas en varios equipos, y todo debe ser unificado para que parezca parte de una sóla aplicación.
Sofia provee la metodología que puede reemplazar o complementar a las hojas de estilo en HTML. Esta metodología es simple y funciona en todo navegador, basándose las propiedades en archivos de texto conteniendo pares de nombre-valor para características de la página, como fondo, tipos de letra.

PROPIEDADES

Ruta del archivo de propiedades

Antes de que cualquier aplicación Sofia pueda ejecutarse, el servidor de aplicaciones (y la máquina virtual) debe recibir la ruta, en el sistema de archivos, en el cual se encuentran los archivos de propiedades. Esto se hace con el switch –D en la línea de comando al levantar el servicio. Ej.:
-Dsalmon.props.path="C:\Desarrollo\Sofia2-2\salmonprops"

Organización de los archivos de propiedades

El principal archivo es *System.properties* . Este contiene el valor predeterminado para todas las páginas construidas con Sofia. Además, las páginas pueden ser agrupadas por aplicación, en cuyo caso los valores predeterminados son leídos de un archivo con otro nombre, *test.properties* si la aplicación se llama “test”. Si dentro del archivo de propiedades de la aplicación no encuentra la propiedad, entonces luego busca en *System.properties*.
No es lo mismo una aplicación Sofia que una aplicación web. Una aplicación Sofia se ejecuta en el contexto de una aplicación web. Por conveniencia, se debería escoger el mismo nombre para no confundir.

Acceso a las propiedades

Las clases para acceder a las propiedades están en el paquete `com.salmonllc.properties.`

```
Props p = getSystemProps();
String s = p.getProperty("nombre_de_propiedad");
```

Para obtener propiedades de una aplicación específica:

```
Props p =
getSystemProps("nombre_de_aplicacion",null);
String s = p.getProperty("nombre_de_propiedad");
```

Creando un ejemplo

El patrón de diseño utilizado por Sofia es MVC.

La parte de la “vista” en Sofia está implementada como páginas jsp con tags personalizados (custom tags).

Vamos a crear una aplicación sofia que se llame “test”.

1. Asegurarse que la ruta al archivo de propiedades sea correcta, y que se encuentre System.properties y, de ser necesario, crear test.properties.
2. La librería de tags de Sofia (taglib.tld) debe estar disponible para la aplicación. Puede ser puesta en el directorio WEB-INF. Luego se la incluye en el archivo jsp (uri="/taglib.tld"). Esta es la misma ruta con que se declara el taglib dentro de `web.xml`.
3. Debe incluir *salmon.jar* en la aplicación web, dentro de *WEB-INF/lib*.
4. Escribir una página jsp:

```
<%@page extends="com.salmonllc.jsp.JspServlet" %>
<%@taglib uri="/taglib.tld" prefix="salmon" %>
<html>
     <salmon:page application="demo"/>
     <salmon:body/>
     <salmon:text name="text1" text="Hello World"
font="DefaultFont"/>
     <salmon:endbody/>
     <salmon:endpage/>
</html>
```

La primera línea indica que la página extiende de `com.salmonllc.jsp.JspServlet`. Esto no es estrictamente necesario, pero ayuda a Sofia a que el procesamiento de la página sea más eficiente.
Luego se referencia a la librería de tags, se abre la página html.
Antes de cualquier otro tag de Sofia debe estar el tag `<salmon:page>`.
Después el tag `<salmon:body>`, que reemplaza al tag <body> de html.
Dentro del cuerpo se desarrolla la página y al final se cierran los tags.

Escribiendo GUI en Java

Además de escribir componentes visuales en HTML, es posible escribirlos en Java, generalmente dentro de los controladores de cada página. Esto es útil en caso que se quiera crear componentes que se carguen dinámicamente, por ejemplo.
Sofia provee un conjunto de clases para hacer esto en el paquete `com.salmonllc.html.`
Además de crear componentes como botones, listas, imágenes, y otros, estos componentes pueden ser agrupados dentro de los que se llaman

`contenedores`. Los contenedores pueden ser HtmlContainer (para mostrar y ocultar componentes como un grupo), HtmlTable (agrupar componentes en una tabla y especificar su posición), HtmlDataTable (
UserGuide.pdf página 20

Otro método es crear tags personalizados. Sofia permite hacer esto haciendo que la clase herede directamente de `com.salmonllc.tags.BaseEmptyTag`. La clase representa el componente visuale. Aunque esta forma no es obligatoria ya que se puede implementar como una extensión de tags de la forma común: que la clase herede de `javax.servlet.jsp.tagext.TagSupport` e implemente una interface como `javax.servlet.jsp.tagext.Tag` .
Luego de crear el tag personalizado dentro de la librería, se la debe incluir en un descriptor de librería, como J2EE lo requiere. El descriptor lleva la extensión `.tld` y el directorio es `WEB-INF/tlds/` , aunque no es obligatorio.
En web.xml se debe referenciar a la librería de tags con el nombre de la librería y la ruta y nombre del archivo descriptor.

```
     <taglib>
          <taglib-uri>taglib.tld</taglib-uri>
          <taglib-location>tlds/taglib.tld</taglib-
location>
     </taglib>
```

El controlador

El controlador es el encargado de interpretar las acciones del usuario, generar los cambios en el modelo de la aplicación y mostrar la vista, ya sea refrescando la misma página o redireccionando a otra. Visto de otra forma, es quien integra la vista y el modelo de la aplicación, y como tal lo óptimo es que delegue la mayor parte del trabajo a los componentes de negocio adecuado.
Por lo general se ejecuta cuando se presiona un botón `Submit` dentro de una página.
Un controlador extiende de `com.salmonllc.jsp.JspController`.

El código del controlador de una página puede ser generado en el respectivo IDE, por ejemplo en Eclipse con el plug-in de Sofia. En caso de que éste método no se desee, se puede también generar automáticamente haciendo referencia vía web (en el browser) a la página y adicionando un parámetro. Ej.:

http://localhost:8080/eguanaReports/login.jsp?generateCode=1

El parámetro `generateCode` con cualquier valor asociado permite que el código base para el controlador de la página `index.jsp` se presente en el browser.

Luego se debe asociar el controlador a una página. Se puede asociar hasta 2 controladores para un JSP a través de los atributos del tag. Dinámicamente se puede establecer otro controlador.
Generalmente sólo se necesita un controlador por página; en raros casos dos.

```
<salmon:page
controller="com.ers.controllers.login.Login"/>
```

Los controladores son una manera conveniente para almacenar datos que deben persistir de una petición a otra de una página en particular. Los controladores son, como tales, almacenados en la sesión, y Sofia se encarga que se almacenen bajo un nombre distinto para que no haya problemas en la información cuando se acceda de manera múltiple a la página. Si se tiene información que debe persistir entre invocaciones de la misma página, se lo puede hacer fácilmente guardándola en las variables de instancia del controlador de la página.

Interacción del controlador con la vista
Un controlador obtiene un handle a cada componente visual y luego ejecuta el método `initialize()`.

La forma más común es referenciar al controlador desde la página JSP.
Cuando se llama al JSP se instancia el controlador y crea componentes html (`com.salmonllc.html.*`) para cada componente visual en la página con el nombre definido en la misma.
Luego se puede obtener una referencia al componente dentro del método deseado:

```
HtmlSubmitButton b =
(HtmlSubmitButton)getComponent("login");
    b.setDisplayName("Presionado");
```

Eventos
Ya se dijo que el controlador es responsable de interpretar las acciones del usuario sobre la página. Para esto se utilizan los “event listeners”, interfaces destinadas al manejo de eventos que en Sofia se encuentran en el paquete `com.salmonllc.html.events`. Las más utilizadas son `PageListener` y `SubmitListener`.

PageListener contiene 4 eventos que se disparan cuando una página es requerida:

```
package com.salmonllc.html.events;
public interface PageListener extends
java.util.EventListener{
      /**
       * Este evento se dispara cada vez que una
      página es requerida.
       */
       void pageRequested(PageEvent p) throws
      Exception ;
       /**
       * Este evento se dispara cada vez que una
      página es requerida.
       */
       void pageRequestEnd(PageEvent p) throws
      Exception ;
       /**
       * Este evento ocurre cada vez que se hace
      submit.
       */
       void pageSubmitEnd(PageEvent p);
       /**
       * Este evento ocurre cada vez que se hace
      submit.
       */
       void pageSubmitted(PageEvent p);
}
```

SubmitListener contiene 1 evento que se dispara cuando se hace submit a la página:

```
package com.salmonllc.html.events;
public interface SubmitListener extends
java.util.EventListener {
       /**
       * Este evento lo dispara el componente html
      que hizo submit. */
       boolean submitPerformed(SubmitEvent e) throws
      Exception;
}
```

Cualquier clase que implemente estas interfaces puede hacerse cargo de los eventos de una página dada, pero por conveniencia es el controlador quien se encarga siempre de hacerlo. Sin embargo en alguna ocasión se puede necesitar que sea otra clase, no controlador.

Contenedores

Hay veces en que es útil encapsular partes de la página en secciones. Este encapsulamiento se lo hace con contenedores (de componentes html).

Un contenedor puede o no tener su propio controlador. Es decir que utiliza un controlador específico para los eventos de los componentes dentro del contenedor o puede utilizar el mismo controlador de la página en la que se encuentra.

Un contenedor puede ser incluido como tal en varias páginas. También puede hacerse visible o invisible bajo ciertas acciones.

ANEXO 4

BITÁCORA DE DESARROLLO

Una parte del proceso de desarrollo ha ido documentanda para posteriores referencias y con fines educativos. Se podrá ver cómo se han ido presentando y resolviendo los retos en la implementación de Eguana Reports.

Inicio

Primero hay que empezar con lo básico.
Se debe crear la estructura de directorio del proyecto en Eclipse. Cada directorio tiene un README.txt para identificar el objetivo o lo que se debe almacenar.
La aplicación será un módulo web. No habrán EJBs, en su lugar se utiliza Hibernate: una herramienta de mapeo objeto-relacional para representar las tablas en objetos.

SOFIA-DREAMWEAVER

Instalar la extensión de Sofia para Dreamweaver. Dentro de la distribución de Sofia, en la carpeta `Resources/Dreamweaver` se encuentran las extensiones. Instalar Dreamweaver y ejecutar "Macromedia Extension Manager". Agregar la extensión de Sofia que se encuentra en `Resources/Dreamweaver`.
Una vez instalado, al abrir Dreamweaver se debe configurar las opciones de Sofia en el menú `Commands-> Salmon SetProperties`

```
Prefix: salmon:
Application: eguanaReports
Tag Lib: taglib.tld
Servlet Url: http://localhost:8080/eguanaReports
Translator mode: ON
Default Language: en
```

Para que estas propiedades funcionen la aplicación eguanaReports debe declarar los servlets que Dreamweaver utiliza para mostrar la página a medida que se va diseñando. Los servlets se los declara en `WEB-INF/web.xml` de la aplicación:

```
<servlet>
  <servlet-name>Translator</servlet-name>
  <servlet-
class>com.salmonllc.servlets.Translator</servlet-
class>
</servlet>

<servlet>
  <servlet-name>DwInfoTranslator</servlet-name>
  <servlet-
class>com.salmonllc.servlets.DwInfoTranslator</servl
et-class>
</servlet>
<servlet-mapping>
  <servlet-name>Translator</servlet-name>
  <url-pattern>/fw/Translator</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>DwInfoTranslator</servlet-name>
  <url-pattern>/DwInfoTranslator</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>Translator</servlet-name>
  <url-pattern>/Translator</url-pattern>
</servlet-mapping>
```

Se puede definir un Site para el proyecto, en el menú Site-> New Site, o dentro del mismo Commands->Salmon SetProperties-> Servlet URL (clic en la carpeta)-> Data Sources:

```
Site name: eguanaReports
Server Technology: Yes…. JSP
Development: Edit and test locally
Store files:
C:\Desarrollo\eclipse\workspace\EguanaReportsSofia\w
eb\public\
URL: http://localhost:8080/eguanaReports (Hacer
Test)
```

```
Copy files to a remote server? : no

Finalmente escribir las siguientes propiedades en el
salmonprops\System.properties:

IDEBrowserPath=C:\\Archivos de programa\\Internet
Explorer\\IEXPLORE.EXE
IDEBrowserPath.IE=C:\\Archivos de programa\\Internet
Explorer\\IEXPLORE.EXE
IDEFrameworkResourcesPath=C:\\Desarrollo\\Sofia2-
2\\Resources
IDEDefaultHost=localhost:8080
IDEDreamWeaverPath=C:\\Archivos de
programa\\Macromedia\\Dreamweaver MX
JSPEngineDebugInputFile=c:\\Desarrollo\\dreamweaver.
log
JSPEngineDebugOutputFile=c:\\Desarrollo\\dreamweaver
.log
```

Luego hay que crear un archivo build.xml para hacer el deploy automático con ant.

Hay que hacer pruebas con las herramientas y librerías que se van a utilizar en el proyecto.

- Crear la base de datos de Eguana. (docsProyecto/Eguana)
- Probar driver mysql, accesando a una tabla de Eguana.
- Configurar hibernate para consultar una tabla Eguana en mysql (docsProyecto/hibernate.txt).
    - Crear el datasource en Tomcat: conf/server.xml si se usa Tomcat.
    - Crear el datasource en Jboss: leer documentación dentro del proyecto.
    - Crear el archivo de configuración: WEB-INF/classes/hibernate.cfg.xml
- Configurar Sofia para crear una página sencilla.

Punto inicial
Existen pruebas (proyectos) hechas con Hibernate y Sofia, para la funcionalidad más básica, haciendo deploy en Jboss

24/sep/2004

- El esqueleto de la aplicación EguanaReportsSofia está creado.
- Crear el build.xml y build.properties
- Incluir las librerías básicas para Hibernate, JasperReports y Sofia en WEB-INF/lib
- Crear index.jsp utilizando Dreamweaver, con la extensión de Sofia, e importarlo al proyecto de Eclipse.
- Crear un war, aunque no exista la funcionalidad, para verificar hasta ahora que la estructura esté bien construida.
- Hacer deploy en Jboss y levantar login.jsp (`http://localhost:8080/eguanaReports/`) (no levantó).
- Corregir web.xml: que la página de error y la de bienvenida existan.

```
     <welcome-file-list>
          <welcome-file>/login.jsp</welcome-file>
     </welcome-file-list>
     <error-page>
          <error-code>404</error-code>
          <location>/error.jsp</location>
     </error-page>
```

- Incluir taglib.tld de Sofia en el directorio WEB-INF.
- El taglib debe estar declarado en web.xml, de tal forma que el URI sea el que se incluye en el tag dentro de la página JSP.

```
  <taglib>
    <taglib-uri>/taglib.tld</taglib-uri>
    <taglib-location>taglib.tld</taglib-location>
  </taglib>
```

- Corregir las páginas JSP para incluir correctamente el taglib, según el URI declarado en web.xml.

```
<%@ taglib uri="/taglib.tld" prefix="salmon"%>
<%@ page extends="com.salmonllc.jsp.JspServlet"%>
<html>
<salmon:page/>. . . . . . . . etc.
```

- Además, incluir en el web.xml el servlet `SalmonPropsPath` para que apunte al correcto directorio de “`salmonprops`”.

```
<servlet>
     <servlet-name>SalmonPropsPath</servlet-name>
<servlet-class>
com.salmonllc.servlets.SalmonPropsPath</servlet-
class>
     <init-param>
          <param-name>salmon.props.path</param-name>
          <param-value>
C:\Desarrollo\Sofia2-2\salmonprops</param-value>
     </init-param>
     <load-on-startup>1</load-on-startup>
```

```
</servlet>
```

- Levantar de nuevo la página (ahora sí funcionó).
- El siguiente paso es crear el controlador para la página. Sofia define un controlador por cada JSP (¡un controlador por cada JSP!). El plug-in de Sofia en Eclipse es, al menos por ahora, una complicación para hacerlo funcionar. Así que utilizaremos la otra forma de generar el controlador para `login.jsp`, llamando vía web a la misma página, pero con un parámetro adicional, de la siguiente forma:
  http://localhost:8080/eguanaReports/login.jsp?generateCode=true

  Modificándo el código un poco se crea el controlador.

27/sep/2004
- Referenciar correctamente al controlador dentro de la página JSP.
  ```
  <salmon:page
  controller="com.ers.controllers.login.Login"/>
  ```
- Leer manual de Sofia (UserGuide.pdf).

28/sep/2004
- Leer tutorial de implementación de tags, para comprender Sofia.
- Mover taglib.tld de Sofia a WEB-INF/tlds/ y hacer los respectivos cambios a `web.xml`. y a los JSPs donde se haga referencia a esta librería de tags.
  ```
  <taglib>
       <taglib-uri>taglib.tld</taglib-uri>
       <taglib-location>tlds/taglib.tld</taglib-
  location>
  </taglib>
  ```

- Continuar documentación de Sofia.

29/sep/2004
- Continuar lectura de manual y documentación de Sofia, para comprender correctamente el funcionamiento de los controladores.
- Revisar código, hacer pruebas. No entiendo el redireccionamiento de páginas.

30/sep/2004
- Continuar lectura de manual y documentación de Sofia, para comprender el modelo.
- Tener en cuenta que el modelo en este caso debe ser congruente entre Sofia y Hibernate. Aparentemente las clases persistentes de

Hibernate pueden funcionar perfectamente como modelo de datos en Sofia.

- Antes de continuar con el modelo, generar la base de datos y una tabla inicial (al menos) para el proyecto EguanaReports.
  Nombre de la base: EguanaReports
  Tabla a crear: usuario

```
CREATE TABLE `usuario` (
  `usuario_id` int(11) NOT NULL auto_increment,
  `login` varchar(20) NOT NULL default '',
  `password` varchar(20) NOT NULL default '',
  `administrador` tinyint(1) NOT NULL default '0',
  `nombre` varchar(100) NOT NULL default '',
  `email` varchar(100) NOT NULL default '',
  PRIMARY KEY  (`usuario_id`)
) TYPE=MyISAM;

INSERT INTO `usuario`
 (`usuario_id`, `login`, `password`,
`esAdministrador`, `nombre`, `email`)
VALUES (NULL, 'admin', 'admin', 1, 'Administrador,
'admin@eguanareports.com);
```

- Crear un usuario en Mysql con permisos para la base de datos EguanaReports, con Mysql Control Center o vía comando:

```
mysql> GRANT ALL PRIVILEGES ON EguanaReports.* TO
eguanareports@localhost IDENTIFIED BY
'eguanareports';
```

o

```
mysql> GRANT ALL PRIVILEGES ON EguanaReports.* TO
eguanareports@% IDENTIFIED BY 'eguanareports';
```

...donde % significa cualquiera.

```
Crear el datasource en JBoss. Archivo eguanareports-
ds.xml en JBOSS_HOME/server/default/deploy:
<?xml version="1.0" encoding="UTF-8"?>

<datasources>
       <local-tx-datasource>
         <jndi-name>EguanaReportsDS</jndi-name>
          <connection-url>
jdbc:mysql://localhost/EguanaReports
    </connection-url>
```

```
         <driver-
class>com.mysql.jdbc.Driver</driver-class>
         <user-name>eguanareports</user-name>
         <password>eguanareports</password>
       </local-tx-datasource>
</datasources>
```

Crear hibernate.cfg.xml dentro del proyecto, en el directorio WEB-INF/classes:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration
PUBLIC "-//Hibernate/Hibernate Configuration
DTD//EN"
"http://hibernate.sourceforge.net/hibernate-
configuration-2.0.dtd">

<hibernate-configuration>
     <!-- SessionFactory es el concepto Hibernate
para UNA fuente de datos -->
     <session-factory>
          <!-- Utilizar el JNDI declarado en
JBOSS_HOME/server/default/deploy/eguanareports-
ds.xml -->
<property name="connection.datasource">
java:/EguanaReportsDS
</property>
          <property name="show_sql">false</property>
          <property name="dialect">
net.sf.hibernate.dialect.MySQLDialect
</property>
          <!-- Mapping files -->
          <mapping
resource="com/ers/model/hibernate/Usuario.hbm.xml"/>
     </session-factory>
</hibernate-configuration>
```

El nombre JNDI para la fuente de datos es EguanaReports
El atributo `show_sql` en trae sirve para ver en bajo nivel las operaciones y sentencias sql que realiza Hibernate.
El archivo de mapeo (objeto-relacional) para la tabla usuario es `Usuario.hbm.xml`, que aún no ha sido creado.
En particular se debe tener en cuenta, según el dtd, que el archivo de configuración puede tener 0 o más elementos `property`, y debe tener

al menos 1 elemento `mapping` (o sea referencia a un archivo de mapeo de clase persistente).

2/oct/2004

Hay que probar qué sucede con `SessionFactory` cuando hay dos aplicaciones Hibernate en el mismo servidor.
Respuesta:
Si existen 2 aplicaciones Hibernate en el mismo contenedor, cada aplicación tiene su propio archivo de configuración de Hibernate (`hibernate.cfg.xml`).
Lo que se ha visto hasta el momento es que ambos archivos de configuración deben estar correctamente escritos. Si uno está bien, y el otro no, simplemente las 2 aplicaciones darán error.
Dentro de la configuración se declara `session-factory`, pero Hibernate no sabe cómo identificar qué `session-factory` pertenece a qué aplicación, entonces es probable que salga una excepción "`No persister for class ...`" por haber escogido la configuración equivocada.
No encuentro documentación al respecto. Intento poniendo el atributo `name` en `session-factory`, pero no funciona.

- Crear la clase persistente `Usuario.java`

```
package com.ers.model.hibernate;

/**
 * Clase para persistencia con Hibernate,
 * representa a la tabla usuario en la base de datos
EguanaReports
 *
 */
public class Usuario {
     private int id;
     private String login; // login name del usuario
de reportes
     private String password;
     private boolean administrador; // true si es
administrador de reportes, false en caso contrario
     private String nombre; // nombre completo del
usuario de reportes.
     private String email;

     public Usuario(){
     }
     public void setId(int id){
          this.id = id;
```

```
	}
	public void setLogin(String login){
		this.login = login;
	}
	public void setPassword(String password){
		this.password = password;
	}
	public void setAdministrador(int
administrador){
		this.administrador = administrador;
	}
	public void setNombre(String nombre){
		this.nombre = nombre;
	}
	public void setEmail(String email){
		this.email = email;
	}
	public int getId(){
		return id;
	}
	public String getLogin(){
		return login;
	}
	public String getPassword(){
		return password;
	}
	public boolean getAdministrador(){
		return administrador;
	}
	public String getNombre(){
		return nombre;
	}
	public String getEmail(){
		return email;
	}

}
```

- Escribir el archivo de mapeo `Usuario.hbm.xml`, a la misma altura que la clase `Usuario`. El archivo de mapeo debe estar relacionado en `hibernate.cfg.xml`.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-
//Hibernate/Hibernate Mapping DTD//EN"
```

```
"http://hibernate.sourceforge.net/hibernate-mapping-
2.0.dtd">

<hibernate-mapping>
     <class name="com.ers.model.hibernate.Usuario"
table="usuario">
          <id name="id" type="int" unsaved-
value="null" >
               <column name="usuarioId" sql-
type="int(11)" not-null="true"/>
               <generator class="increment"/>
          </id>
          <property name="login">
               <column name="login" length="20" not-
null="true"/>
          </property>
          <property name="password">
               <column name="password" length="20"
not-null="true"/>
          </property>
          <property name="administrador">
               <column name="administrador" not-
null="true"/>
          </property>
          <property name="nombre">
               <column name="nombre" length="100"/>
          </property>
          <property name="email">
               <column name="email" length="100"/>
          </property>
     </class>
</hibernate-mapping>
```

- Corregir el archivo `build.xml` para incluir los archivos de mapeo de hibernate en la aplicación final. Esto es porque los archivos de mapeo están en la misma ubicación de las clases de persistencia, lo que no es una ubicación común para un archivo xml.

```
<?xml version="1.0" encoding="UTF-8"?>

<project basedir="." default="deploy-jboss"
name="EguanaReportsSofia">
  <!-- Propiedades globales -->
  <property file="${basedir}\build.properties" />
  <property name="deploy.tomcat.dir"
value="${TOMCAT_HOME}/webapps"></property>
```

```
  <property name="deploy.jboss.dir"
value="${JBOSS_HOME}/server/default/deploy"></proper
ty>

  <target name="init">
    <property name="app.name"
value="eguanaReports"/>

    <property name="web.src.dir"
value="./web/src" />
    <property name="web.docroot.dir"
value="./web/public" />
    <property name="web.descriptor.dir"
value="./web/WEB-INF" />
    <property name="lib.dir"          value="./lib" />
<!-- Las librerias finales -->
    <property name="build.dir"        value="./build"
/> <!-- El directorio temporal antes de construir el
modulo -->
    <property name="class.files"      value="*.class"
/>
    <property name="java.files"       value="*.java"
/>
    <property name="xml.files"        value="*.xml" />

  </target>

  <path id="classpath">
    <pathelement
location="${TOMCAT_HOME}/common/lib/servlet-
api.jar"/>
    <pathelement
location="${target.webinf}/classes"/>
  </path>

  <!-- Para incluir en el classpath cualquier
librería, que se agregue al directorio WEB-INF/lib
de la aplicación -->
  <fileset id="applicationJarLibFile"
dir="./web/WEB-INF/lib">
     <include name="**/*.jar"/>
  </fileset>
  <path id="applicationJarLib">
     <fileset refid="applicationJarLibFile" />
  </path>
```

```
  <!-- Borrar el directorio de compilacion. Primero
setear variables de proyecto. -->
  <target name="clean" depends="init">
     <delete dir="${build.dir}" /> <!-- directorio
con todo compilado -->
     <!--<delete dir="${lib.dir}" /> <!- Directorio
con los empaquetados finales del proyecto -->
  </target>

   <!-- Crear los directorios necesarios para crear
el EAR. Estos directorios deben
          ser borrados una vez que el ear sea
creado. -->
   <target name="setup" depends="clean">
      <mkdir dir="${lib.dir}" />
      <mkdir dir="${build.dir}" />
      <mkdir dir="${build.dir}/web" />
      <mkdir dir="${build.dir}/web/WEB-INF" />
      <mkdir dir="${build.dir}/web/jsps" />
      <mkdir dir="${build.dir}/web/images" />
      <mkdir dir="${build.dir}/web/WEB-INF/classes"
/>
      <mkdir dir="${build.dir}/web/WEB-INF/lib" />
   </target>

   <!-- Compilar las clases Web (.java) y copiar
archivos de mapeo de clases persistentes -->
   <target name="web-classes" >
      <javac srcdir="${web.src.dir}"
             destdir="${build.dir}/web/WEB-
INF/classes"
             debug="on">
          <classpath refid="classpath"/>
          <classpath refid="applicationJarLib"/>
      </javac>
       <!-- Copiar los archivos de mapeo de las
clases persistentes de Hibernate
             Estos archivos están en el mismo
directorio que las clases persistentes.-->
       <copy todir="${build.dir}/web/WEB-
INF/classes">
          <fileset dir="${web.src.dir}/"
includes="**/*.xml"/>
       </copy>
```

```
   </target>

   <target name="web-view">
      <!-- Copiar los JSP al directorio build -->
      <copy todir="${build.dir}/web/">
         <fileset dir="${web.docroot.dir}/"
includes="**"/>
      </copy>
      <!-- Copiar el descriptor WEB al directorio
build -->
       <copy todir="${build.dir}/web/WEB-INF">
          <fileset dir="${web.descriptor.dir}"
includes="**"/>
       </copy>
   </target>

   <!-- Crear el war -->
   <target name="war" depends="setup,web-view,web-
classes">
      <jar jarfile="${lib.dir}/${app.name}.war">
        <fileset dir="${build.dir}/web"
includes="**" />
      </jar>
   </target>

   <target name="deploy-tomcat" depends="war">
        <!-- Hacer deploy en TOMCAT es copiar el war
en webapps -->
        <copy todir="${deploy.tomcat.dir}">
           <fileset dir="${lib.dir}"
includes="${app.name}.war"/>
        </copy>
   </target>

   <!-- Hacer deploy en JBOSS, configuración
default, es copiar el war en /deploy -->
   <target name="deploy-jboss" depends="war">
       <copy todir="${deploy.jboss.dir}">
          <fileset dir="${lib.dir}"
includes="${app.name}.war"/>
       </copy>
   </target>
</project>
```

- Crear tabla, configuración de Hibernate, clase persistente, archivo de mapeo para Grupo, Datasource. Esto por ahora. Hacer pruebas de almacenamiento.

4/oct/2004

- Aún no está claro cómo cambiar de página en un controlador de Sofia. El siguiente paso es captar parámetros (página `login.jsp`) y hacer conexión con la base a través de Hibernate.
- Para cambiar de página en Sofia, dentro de un controlador, en este caso dentro del método submitPerfomed():

```
     public boolean submitPerformed(SubmitEvent
event) {
          Usuario usuario = null;
     try {
          // Averiguar si el botón presionado fue el
de Login
          if (event.getSource() == _login){
               //Verificar usuario
               AdministradorLogin administradorLogin
= new AdministradorLogin();
               if ((usuario =
administradorLogin.esUsuarioValido(_user.getValue(),
_password.getValue())) != null){
                    // Guardar objeto en la sesión

     getCurrentRequest().getSession().setAttribute("
login.usuario",usuario);
                    //Redireccionar página en el
browser

     sendRedirect(applicationName+"/index.jsp");
               } else

     sendRedirect(applicationName+"/error.jsp");
               }
          } catch (IOException ioe) {
               System.out.println("Controlador Login
- submitPerformed :"+ioe);
          } catch (HibernateException he) {
               System.out.println("Controlador Login
- submitPerformed :"+he);
     }
          return true;
     }
```

… donde `applicationName = getCurrentRequest().getContextPath()`. En este caso `applicationName = "/eguanaReports"`. Para `sendRedirect()` se debe utilizar el nombre del módulo web más el path completo al recurso (jsp) dentro del módulo.

- Y el método `administradorLogin.esUsuarioValido` es:

```
     public Usuario esUsuarioValido(String login,
String password){
          Usuario usuario = null;
     try {
          // La sesión es un objeto que representa
una unidad de trabajo con la base.
          session = getSession();
          // Busca el usuario en la base
          //usuario = (Usuario) session.find("from
Usuario as usuario where usuario.login =
'fperez'",login, Hibernate.STRING).get(0);
          usuarios = session.find("from Usuario as
usuario where usuario.login = ? and usuario.password
= ?",
                         new Object[] {login,
password},
                         new Type[]
{Hibernate.STRING, Hibernate.STRING});

          // Si existe y el password es correcto,
retorna el objeto Usuario
          if (usuarios.size() > 0){// &&
(usuario.getPassword() == password)){
               session.close();
               return (Usuario) usuarios.get(0);
          }

     } catch (HibernateException he) {
          System.out.println("AdministradorLogin -
esUsuarioValido :"+he);
     } finally {
          try {
               session.close();
          } catch (HibernateException he) {

     System.out.println("AdministradorLogin -
esUsuarioValido :"+he);
          }
```

```
        }
            return null;
        }
```

5/oct/2004

- Leer cómo manipular datos en Hibernate, para traer y guardar objetos.
- He probado, pero los datos no se cargan en la clase persistente.

6/oct/2004

- Configurar log4j, por ahora para mensajes de error y mensajes de consola.
- Debe `estar log4j-1.2.8.jar` en `WEB-INF/lib`.
- Crear `log4j.properties` en `WEB-INF/classes` :

```
### to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppend
er
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.Patter
nLayout
log4j.appender.stdout.layout.ConversionPattern=%d{AB
SOLUTE} %5p %c{1} - %m%n
### set log levels - for more verbose logging change
'info' to 'debug' ##
#log4j.rootLogger=error, stdout, file
log4j.rootLogger=error, stdout

log4j.logger.net.sf.hibernate=info
```

- Ya funciona el acceso con Hibernate, a partir de una página Sofia.
- Implementar ahora menú principal con el tag `navvar` de Sofia
- La computadora está demasiado lenta. Falta de memoria.

7/oct/2004

- Implementar un menú con Sofia. No hay documentación suficiente.

8/oct/2004

- El menú básico está creado. Es parte de un contenedor de Sofia, con su propio controlador. De esta forma puede ser incluido en cualquier página sin ningún problema.
- Una página cualquiera incluye el archivo jsp donde se encuentra el contenedor. Se incluye dentro del tag `<salmon:body/>`

```
<%@page extends="com.salmonllc.jsp.JspServlet" %>
<%@taglib uri="taglib.tld" prefix="salmon" %>
<html>
```

```
     <salmon:page
controller="com.ers.controllers.ControladorIndex"/>
     <salmon:body/>
               <%@ include file="cabecera.jsp"%>
     <salmon:endbody/>
     <salmon:endpage/>
</html>
```

Debe usarse include file, y no include page:

```
     <jsp:include page="cabecera.jsp"></jsp:include>
```

- El archivo `cabecera.jsp` :

```
<salmon:container name="menuPrincipalContainer"
containerclass="com.ers.controllers.ControladorMenuP
rincipal">
     <salmon:navbar name="menuPrincipal"
bgcolor="#CCCCCC"  selectedmarkerimage=""
textimage="" vspaceimage=""
hspaceimage="images/null.gif">
          <salmon:navbargroup name="eguanaReports"/>
               <salmon:navbaritem name="usuario"
title="Usuarios" submenuname="usuario" />
                    <salmon:navbarsubitem
name="nuevoUsuario" title="Nuevo"
href="usuario/nuevo.jsp" submenugroup="usuario" />
                    <salmon:navbarsubitem
name="listadoUsuario" title="Listado"
href="usuario/listado.jsp" submenugroup="usuario" />
               <salmon:navbaritem name="datasource"
title="Datasources" submenuname="datasource" />
                    <salmon:navbarsubitem
name="nuevoDatasource" title="Nuevo"
href="datasource/nuevo.jsp"
submenugroup="datasource" />
                    <salmon:navbarsubitem
name="listadoDatasource" title="Listado"
href="datasource/listado.jsp"
submenugroup="datasource" />
     </salmon:navbar>
</salmon:container>
```

- Las propiedades para el menú se definen de manera general en el archivo `.properties` (`eguanaReports.properties` en este caso):

```
NavBarBgColor=white
NavBarAlign=
NavBarBorder=
NavBarCellPadding=4
NavBarCellSpacing=1
NavBarHSpace=10
NavBarVSpace=0
NavBarHeight=
NavBarWidth=172
NavBarHrefStyle=font-family: Verdana, Arial,
Helvetica, sans-serif; font-size: 10px; text-
decoration: none; font-weight:bold; COLOR: #FFFFFF;
NavBarHoverStyle=font-family: Verdana, Arial,
Helvetica, sans-serif; font-size: 10px; text-
decoration: none; font-weight:bold;COLOR: #FFCC66;
NavBarTextStyle=font-family: Verdana, Arial,
Helvetica, sans-serif; font-size: 10px; text-
decoration: none; font-weight:bold; color: #FFFFFF;
NavBarSelectedStyle=font-family: Verdana, Arial,
Helvetica, sans-serif; font-size: 10px; text-
decoration: none; font-weight:bold;COLOR: #FFFFFF;
NavBarSelectedHoverStyle=font-family: Verdana,
Arial, Helvetica, sans-serif; font-size: 10px; text-
decoration: none; font-weight:bold;COLOR: #FFCC66;
NavBarShowPopupStyle=showpopupstyle=font-family:
Verdana, Arial, Helvetica, sans-serif; font-size:
10px; text-decoration: none;COLOR: #FFFFFF;
NavBarCellBgColor=#666699
NavBarGroupCellBgColor=#313163
NavBarGroupHoverBgColor=
NavBarSelectedBgColor=#9999CC
NavBarSelectedHoverBgColor=
NavBarShowPopupBgColor=#9999CC
NavBarMarkerImage=images/null.gif
NavBarSelectedMarkerImage=images/null.gif
NavBarTextImage=images/null.gif
NavBarHSpaceImage=images/null.gif
```

Por alguna razón no toma el atributo `NavBarHSpaceImage` y hay que establecerlo en el jsp mismo.

- El controlador para el menú, declarado en el atributo `containerclass` del tag `<salmon:container>`:

```
package com.ers.controllers;
```

```
import com.salmonllc.html.HtmlPage;
import com.salmonllc.jsp.JspContainer;

/**
 * Controlador para el container del menú principal.
 * name="menuPrincipalContainer" en /cabecera.jsp
 */
public class ControladorMenuPrincipal extends
JspContainer {
    public com.salmonllc.jsp.JspNavBar
_menuPrincipal;
     public ControladorMenuPrincipal(String name,
HtmlPage page) {
          super(name, page);
     }

     public void initialize() {
          // Propiedades del menú.
          _menuPrincipal.setHorizontalMode(true);
     }
}
```

9/oct/2004

- Crear página jsp, controlador y clase `AdministradorUsuario` para crear usuario.

11/oct/2004

- Crear clase de ayuda de Hibernate para mantener una instancia de SessionFactory y manejar sesiones con más facilidad. (`HibernateHelper`).

12-13-14/oct/2004

- Diseñar otras páginas.
- Aprender a utilizar las tablas de Sofia con información traída de la base.

```
<salmon:datasource name="listaUsuarios" type="MODEL"
model="com.ers.model.hibernate.Usuario"
autoretrieve="Never">
</salmon:datasource>
```

Primero se declara un datasource en la página. Los datos se almacenarán en objetos de tipo `Usuario` (beans), que se los traerá usando Hibernate de

forma programática en el controlador, por lo tanto se requiere que no se intente la lectura automática (`autoretrieve="Never"`). Significa que `listaUsuarios` va a contener una lista de objetos de tipo `Usuario`.
Más adelante en la misma página se crea un `datatable` haciendo referencia al `datasource` (`listaUsuarios`), donde las filas (`datatablerows`) serán llenadas automáticamente con el contenido de la lista. Lo que se requiere simplemente es hacer referencia (ej.: `listaUsuarios:login`) al atributo (login) del objeto `Usuario` que se almacena en `listaUsuario`.

```
<salmon:datatable name="tablaListaUsuarios"
rowsperpage="10" width="100%" align="Left"
datasource="listaUsuarios">
<salmon:datatableheader>
     <salmon:tr>
          <salmon:td valign="top"
width="10%"></salmon:td>
          <salmon:td valign="top"><salmon:font
type="TableHeadingFont">Login</salmon:font></salmon:
td>
          <salmon:td valign="top"><salmon:font
type="TableHeadingFont">Nombre</salmon:font></salmon
:td>
          <salmon:td valign="top"><salmon:font
type="TableHeadingFont">Email</salmon:font></salmon:
td>
     </salmon:tr>
</salmon:datatableheader><salmon:datatablerows>
     <salmon:tr>
                         <salmon:td
align="Center"><salmon:input type="checkbox"
name="delete"
checkedtruevalue="1"></salmon:input></salmon:td>
          <salmon:td><salmon:text name="filaLogin"
text=""
datasource="listaUsuarios:login”></salmon:td>
          <salmon:td><salmon:text name="filaNombre"
text=""
datasource="listaUsuarios:nombre"/></salmon:td>
          <salmon:td><salmon:text name="filaEmail"
text=""
datasource="listaUsuarios:email"/></salmon:td>
     </salmon:tr>
```

```
</salmon:datatablerows></salmon:datatable>
```

La página jsp se carga cada vez que se referencia (por url, por submit, post) a ésta ya sea desde la misma página o desde otra. El método (del controlador) para llevar a cabo el llenado del contenido de la página es `pageRequested()`.
Una manera simple de llenar la tabla anterior es:

```
          public void pageRequested(PageEvent event) {
               // Llenar la lista de usuarios
               AdministradorUsuarios
     administradorUsuarios = new AdministradorUsuarios();
               // AdministradorUsuario consulta con la
     capa de persistencia los usuarios
               // y devuelve una lista (List con objetos
     tipo Usuario)
               // _listaUsuarios toma la lista según el
     modelo definido en la página jsp: model="...Usuario"
               // y llena la tabla.
               _listaUsuarios.reset(); //reset encera la
     tabla. No siempre es conveniente.

          _listaUsuarios.insertRows(administradorUsuarios
     .listadoTodosUsuarios());
          }
```

… donde `_listaUsuarios.insertRows(Collection c)` inserta automáticamente los datos en la tabla a partir de una colección de objetos (`List extends Collection`), devuelto por el método:

```
          public List listadoTodosUsuarios(){
               try {
                    // La sesión es un objeto que
     representa una unidad de trabajo con la base.
                    session =
     HibernateHelper.getSession();
                    // Consulta la lista de usuarios
                    usuarios = session.createQuery("from
     Usuario usuario order by usuario.login").list();

                    // Si existe retorna la lista de
     usuarios
                    if (usuarios.size() > 0){
                         HibernateHelper.closeSession();
                         return usuarios;
```

```
                }

            } catch (HibernateException he) {

        System.out.println("AdministradorUsuarios -
listadoTodosUsuarios :"+he);
            } finally {
                try {
                    HibernateHelper.closeSession();
                } catch (HibernateException he) {

        System.out.println("AdministradorUsuarios -
listadoTodosUsuarios :"+he);
                }
            }
            return null;
        }
```

16-18/oct/2004

- Analizar tablas de OpenReports. Dado que EguanaReports se basa en OpenReports, la estructura de tablas será básicamente la misma. Al menos por ahora las modificaciones son mínimas.
- Crear el resto de clases persistentes y los archivos de mapeo. No olvidarse de hacer referencia a los archivos de mapeo en `hibernate.cfg.xml`.
- Investigar implementación de relaciones en Hibernate (`many-to-many`, `many-to-one`, y otras)
- Hacer pruebas para validar la configuración.
- Bueno, las pruebas no salen.

19-23/oct/2004

- Aparentemente los errores se deben a mala configuración de mapeo, especialmente para las asociaciones entre entidades. Lo que significa esto es simple: “lee todo el manual”.
- Comprender los conceptos de mapeo (capítulo 5). Distinguir entre ENTIDADES y VALORES.
- Lectura, comprensión y pruebas de persistencia de entidades (clases persistentes). Los capítulos más útiles del manual de hibernate (`hibernate_reference.pdf`) son el 6 (Collection Mapping), el 9 (Manipulating Persistent Data) y el 16 (ejemplos).
- Todo esto para comprender el ciclo de persistencia. Claro que a estas alturas se debe tener buen manejo de las cosas básicas de Hibernate

(y haber leído al menos los capítulos 2, 3, 4, 5 y 11). Y claro, tener en cuenta el DTD para las configuraciones.

- Las entidades borradas de una relación (collección) no se eliminan automáticamente, al menos que se ponga `cascade="all"` (o el método específico de cascada) en el archivo de mapeo para dicha relación.
- Por otro lado, las relaciones bidireccionales deben ser `readonly` en alguno de sus extremos. Esto se logra poniendo `inverse="true"` en el archivo de mapeo para la relación.
- Una relación padre-hijo entre entidades se puede implementar utilizando `one-to-many` bidireccional. Ver capítulo de ejemplo.
- POR OTRO LADO: Crear el esqueleto para lo que será el servicio de reportes de la aplicación. Incluir las librerías necesarias para el funcionamiento de JasperReports:

  JasperReports (jasperreports-0.6.1.jar) necesita:
  - o XML Parser: JAXP 1.1
  - o Jakarta Commons Digester (commons-digester-1.3.jar)
  - o Jakarta Commons BeanUtils (commons-beanutils-1.6.1.jar)
  - o Jakarta Commons Collections (commons-collections-2.1.jar)
  - o Jakarta Commons Logging (commons-logging-1.0.3.jar)
  - o JDBC 2.0 Driver (jdbc2_0-stdext.jar)
  - o iText - Free Java-PDF library (itext-1.01.jar)
  - o XLS: Jakarta POI (poi-2.0-final-20040126.jar)

  Driver Mysql (mysql-connector-java.jar)
- Crear com.ers.servlet.ReporteServlet.
- Declararlo en WEB-INF/web.xml:

```
     <servlet>
          <servlet-name>ReporteServlet</servlet-
name>
          <servlet-
class>com.ers.servlet.ReporteServlet</servlet-class>
     </servlet>
     <servlet-mapping>
          <servlet-name>ReporteServlet</servlet-
name>
          <url-pattern>/reporte</url-pattern>
     </servlet-mapping>
```

- Incluir los drivers (en `WEB-INF/lib`) para las bases de datos a las cuales se puede proporcionar servicio. Es para obtener la conexión al momento de hacer reportes.
- Crear el directorio en la aplicación web en donde se van a grabar los reportes compilados (`.jasper`): a la misma altura que los jsp, en el

directorio `reportes`, de tal forma que se accedan a partir del url base de la aplicación (el contexto).

- Crear un archivo de prueba jsp para simular el link:

```
<a
href="http://127.0.0.1:8080/eguanaReports/reporte?us
uario=prueba&password=123&reporte=reportePrueba&tipo
Salida=PDF">REPORTE PDF</a>
```

24-27/oct/2004

- Al querer grabar varias instancias se genera un error de Objeto Duplicado. En este punto hay que manejar mejor los conceptos de generación de ID para las clases persistentes. Los valores de `unsaved-value="null"` en los archivos de mapeo se los ha reemplazado por `unsaved-value="0"`. De esa forma grabar en "masa" y en cascada no genera problemas.
- Hacer que el ReporteServlet lea los parámetros y los cargue a partir de la base interactuando con Hibernate. Así mismo la conexión vía JDBC o JNDI.

28/oct/2004

- Crear com.ers.servlet.ParametroReporteServlet. Es el que genera código HTML representando los parámetros que hay que ingresar para determinado reporte.
- Declararlo en `WEB-INF/web.xml`.
- Crear pruebas.

19/nov/2004

- Máquina dañada. Datos recuperados
- Un parámetro debe ser llenado por el usuario y pasado al momento de petición de reporte. Si el tipo de parámetro se lo escoge de una lista, implica que la lista tiene valores predefinidos. No hay tabla que refleje esto, por lo tanto se ha creado una nueva tabla llamada "`valorparametro`".
- Se debe crear el mapeo y las relaciones en Hibernate para reflejar los cambios. Esto implica: crear `ValorParametro.java`, `ValorParametro.hbm.xml`, modificar `Parametro.java`, `Parametro.hbm.xml`, y `hibernate.cfg.xml`.

```
root cause
java.lang.NoClassDefFoundError
     com.ers.helper.AdministradorLogin.esUsuarioVali
do(AdministradorLogin.java:64)
```

```
      com.ers.controllers.login.ControladorLogin.subm
itPerformed(ControladorLogin.java:55)
      com.salmonllc.html.HtmlSubmitButton.executeEven
t(HtmlSubmitButton.java:128)
      com.salmonllc.jsp.JspForm.executeEvent(JspForm.
java:382)
      com.salmonllc.html.HtmlContainer.executeEvent(H
tmlContainer.java:79)
      com.salmonllc.html.HtmlPage.processParms(HtmlPa
ge.java:1331)
      com.salmonllc.jsp.JspController.doPost(JspContr
oller.java:605)
      com.salmonllc.jsp.tags.PageTag.doStartTag(PageT
ag.java:218)
      org.apache.jsp.login_jsp._jspx_meth_salmon_page
_0(login_jsp.java:91)
      org.apache.jsp.login_jsp._jspService(login_jsp.
java:50)
      com.salmonllc.jsp.JspServlet.service(JspServlet
.java:313)
      org.apache.jasper.servlet.JspServletWrapper.ser
vice(JspServletWrapper.java:324)
      org.apache.jasper.servlet.JspServlet.serviceJsp
File(JspServlet.java:292)
      org.apache.jasper.servlet.JspServlet.service(Js
pServlet.java:236)
      javax.servlet.http.HttpServlet.service(HttpServ
let.java:810)
```

Error: Parametro.java debe incluir getter y setter para la lista de valores.

23/nov/2004

- Corrigiendo y mejorando la generación de parámetros.

26/nov/2004

- Al cargar la lista de parámetros, y en general cualquier lista de objetos, en una relación bidireccional se cargan elementos nulos en la colección.
- Esto es porque Hibernate no soporta colecciones indexadas en una relación bidireccional de one-to-many o many-to-many.
- CORREGIR: Cambiar todas las colecciones List a Set, o implementar una solución poco agradable que es eliminar todos los elementos null en el setter de la lista.

29/nov/2004

- El manual dice que no se debe implementar listas en relaciones bidireccionales. En una relación many-to-many bidireccional se supone (no dice nada) que debería funcionar, pero no es así.
- En tal caso todas las listas se cambiaron a Set. Si se desea ordenar se utiliza un LinkedHashSet y en el archivo de mapeo se implementa el order-by para hacer la consulta SQL ordenada según el campo de la tabla. No veo otra manera.
- Parece, ahora sí, que todo está correcto. La capa de persistencia está completa.
- Hubo un par de pequeños cambios en la estructura de la base de datos.
- Siguiente paso: diseñar las interfaces de administración.

Este es un ejemplo del proceso de desarrollo. Si documentáramos todo, obviamente, no nos alcanzarían las páginas.

Más adelante nos daríamos cuenta que Sofia no cumplía nuestras necesidades. Limita la libertad al crear una página JSP y existe una relación poco flexible entre la página y el controlador. Struts permite hacerlo de una manera más natural y se combina bien con los elementos HTML y otros componentes dinámicos. Entonces rehicimos el proyecto utilizando Struts.

ANEXO 5

INSTALACIONES ANTERIORES

1. Instalación del J2SDK

Instalación de JAVA → j2sdk-1_4_2_02-windows-i586-p.exe

Para esto los instaladores proveen de un Wizard de instalación fácil, el cual incluye también el Java Run Time Environment ver 1.4.2_02, Java Web Start.

1.1. Se configura las *variables de ambiente del sistema* requerido para Java en el sistema operativo:

- **%JAVA_HOME%** – La ruta del home de Java para definir las rutas relativas que necesiten referenciar a este directorio.
- **%CLASSPATH%** – Se definen las rutas del home y bin de Java.
- **%PATH%** – Se definen las rutas del directorio bin de Java.

1.2. Se verifica que se encuentre la instalación correcta:
C:\>java –version

```
java version "1.4.2_02"
Java(TM) 2 Runtime Environment, Standard Edition
(build 1.4.2_02-b03)
Java HotSpot(TM) Client VM (build 1.4.2_02-b03,
mixed mode)
```

2. Instalación del Apache Tomcat 5.0

Instalación de Tomcat → jakarta-tomcat-5.0.19.exe
Para esto los instaladores proveen de un Wizard de instalación fácil.

2.1. Se configura las variables de ambientes requeridas para TOMCAT:

- **%CATALINA_HOME%** – La ruta del home de Tomcat para definir las rutas relativas que necesiten referenciar a este directorio.
- **%CLASSPATH%** – Se definen la ruta del home de Tomcat.

2.2. Iniciar los servicios del Apache Tomcat.

- Se puede realizar de modo gráfico:
  Start -> Programs -> Apache Tomcat 5.0 -> Start Tomcat
  La salida se puede observar en la consola estilo Windows que se presenta en el icono de Apache Tomcat (Barra de Tareas)
  Open Console Monitor.

- Se puede realizar de en línea comando:
  %CATALINA_HOME%\bin\startup.bat
  Using CATALINA_BASE: E:\jakarta-tomcat-5.0.19
  Using CATALINA_HOME: E:\jakarta-tomcat-5.0.19
  Using CATALINA_TMPDIR: E:\jakarta-tomcat-5.0.19\temp
  Using JAVA_HOME: E:\j2sdk1.4.2_02

  La salida se observa en la consola DOS que se levanta al iniciar los servicios de TOMCAT.

2.3. Verificación de la instalación a través de una conexión vía browser al localhost por el puerto default. http://localhost:8080

2.4. Adicionalmente se pueden ejecutar y observar los códigos de ejemplos: JSP, Servlets y WebDAV

2.5. Se observa una suite de administración de Tomcat, por medio de los links al Tomcat Administrator (Web Server) y Tomcat Manager.

2.6. Para bajar los servicios de Tomcat.

- Se puede realizar de modo gráfico:

En el icono de Apache Tomcat (Barra de Tareas) se escoge la opción *Shutdown: Tomcat5*

- Se puede realizar en línea de comando:

%CATALINA_HOME%\bin\shutdown.bat
Using CATALINA_BASE: E:\jakarta-tomcat-5.0.19
Using CATALINA_HOME: E:\jakarta-tomcat-5.0.19
Using CATALINA_TMPDIR: E:\jakarta-tomcat-5.0.19\temp
Using JAVA_HOME: E:\j2sdk1.4.2_02

3. Instalación de la base de datos MySQL.

Instalación del Motor de MySQL → MySql4.zip
Administrador de la base de datos → MySQL-Front_21_Setup.exe

Para esto los instaladores proveen de un Wizard de instalación fácil.

3.1. Inicializar los servicios de la base de datos winmysqladmin.exe en modo gráfico. Si se realiza en línea de comando se puede levantar de dos maneras:

- Multiusuario - Levantar como servicio
  %MYSQL_HOME%\bin:\> net start mysql

- Monousuario
  %MYSQL_HOME%\bin:\> mysqld --standalone
  %MYSQL_HOME%\bin:\> mysqld-max --standalone
  %MYSQL_HOME%\bin:\> mysqld-max --standalone --debug

- Multiusuario - Detener como servicio
  %MYSQL_HOME%\bin:\> net stop mysql

- Monousuario
  %MYSQL_HOME%\bin:\> mysqladmin -u root shutdown

3.2. Levantar una terminal transaccional a través del archivo:
%MYSQL_HOME%\bin:\> mysql

3.3. Verificación de la conexión por medio del comando:
mysql> status;

3.4. Para bajar los servicios de la base de datos se realiza lo siguiente:

- Multiusuario - Levantar como servicio
  %MYSQL_HOME%\bin:\> net stop mysql

- Monousuario
  %MYSQL_HOME%\bin:\> mysqladmin -u root shutdown

4. Instalación de Eclipse 3.0

Instalación: Desempaquetar → `eclipse-SDK-3.0-win32.zip`

Para iniciar Eclipse hacer doble clic en `%ECLIPSE_HOME%\eclipse.exe`

Revisiones realizadas:

3.1. Construcción de una simple aplicación JAVA

3.2. Creación de SWT Aplicación

3.3. Crear un plug-in Hello World en el menú bar de Eclipse.

El desarrollo de estos 3 ejercicios se realizaró con el menú de ayuda que posee Eclipse de estas 3 aplicaciones.

5. Instalación de SOFIA (Salmon Open Framework for Internet Applications)

Instalación: Desempaquetar → `SOFIAInstallation2-2.zip`

Se trabajó con la siguiente plataforma:

- J2SDK versión 1.4.2 release 02
- Jakarta Tomcat versión 5.0.19
- MySql versión 4.0.20
- Eclipse versión 3.0

Para la revisión de SOFIA se realizó la instalación de un conjunto de aplicaciones de ejemplo de SOFIA sobre la plataforma de TOMCAT, incluyendo la interacción con la base de datos MySql.

Interactuando con la base de datos

5.1 Iniciar los servicios de la base de datos MySQL

```
mysqld --standalone
```

5.2 Realizamos una conexión a la base de datos por medio de una sesión con el usuario administrador (root) y creamos la base sofia.

```
mysql –user=root
mysql> create database sofia;
```

5.3 Realizamos una conexión a la base de datos por medio de una sesión Creamos las estructuras de la base de datos sofia con el script.

```
create_sofia_db_with_data.sql
mysql> source ./create_sofia_db_with_data.sql ;
```

Instalando una aplicación en SOFIA sobre Tomcat

5.4 Copiar el directorio `salmonprops` (`%SOFIA_HOME%`) a `%CATALINA_HOME%`.

5.5 Copiamos los archivos jars desde `%SOFIA_HOME%\Resources\lib` al directorio `%CATALINA_HOME%\common\lib`:

- `Mm.mysql-2.0.11-bin.jar` (MySQL LDBC Driver)
- `Salmon.jar` (SOFIA runtime binary)
- `Portlet-api-1.0.jar` (Portlet API)
- `Activation.jar y mail.jar` (Si no existen copiarlos)

5.6 Copiamos el directorio sofiaExamples (`%SOFIA_HOME%\WebApp`) al directorio `%CATALINA_HOME%\WebApps`.

5.7 Editamos el archivo `sofiaExamples.properties` del directorio `%CATALINA_HOME%\salmonprops`, para modificar lo siguiente:

- Validar los parámetros de conexión a la base de datos (usuario y password).
- JSPSoruce, referenciar el directorio donde se encuentra almacenado el código JSP de los ejemplos de SOFIA
- JavaSource, referenciar el directorio donde se encuentra almacenado el código Java de los ejemplos de SOFIA

Considerar en ambos casos que para ambientes MS-Windows se debe incluir el drive del directorio y el separador debe ser “\\”.

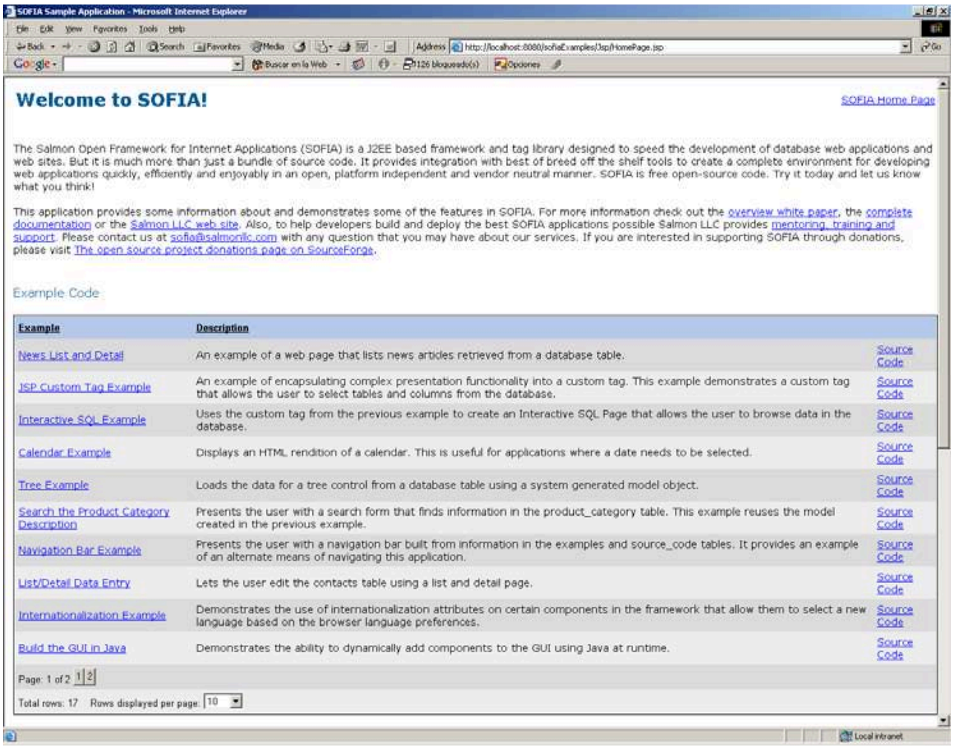
Levantamos los servicios de TOMCAT

5.8 Definimos una variable de ambiente de Sistema %CATALINA_OPTS% con el siguiente valor:
-Dsalmon.props.path=”%CATALINA_HOME%\salmonprops”

Otra alternativa es agregar en las opciones de JAVA Virtual Machine de la configuración del Tomcat Server, la línea:
-Dsalmon.props.path=”%CATALINA_HOME%\salmonprops”

5.9 Para verificar realizamos una conexión mediante el browser http://localhost:8080/sofiaExamples/Jsp/HomePage.jsp

5.10 Finalmente observaremos la página Home de SOFIA con varios ejemplos prácticos para el desarrollo de aplicaciones con este framework.



6. Instalación del Plug-in de SOFIA para Eclipse

Revisiones realizadas:

6.1 La plataforma empleada para esta instalación es Eclipse versión 3.0

6.2 Este punto es opcional.
Copiamos las dos librerías (`SDCLIENT.DLL` y `REGUPDATE.DLL`) que se encuentran en `%SOFIA_HOME%\Resources\DLL` al directorio `%WINDIR%\system32`. Solo para el caso de MS-Windows. Leer ManualInstallationGuide.pdf en `%SOFIA_HOME%` .

6.3 Copiamos el plug-in de Sofia (directorio) `com.salmonllc.eclipse` que está en la ruta `%SOFIA_HOME%\Resources\Eclipse`, al directorio `%ECLIPSE_HOME%\plugins`.

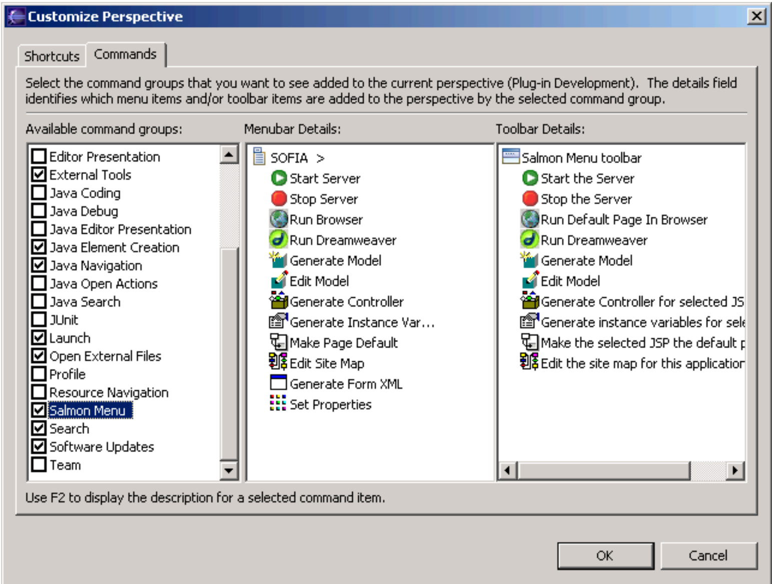
6.4 Si los servicios de TOMCAT están ejecutándose, procedemos a detenerlo. Luego levantaremos el servicio de TOMCAT desde el IDE.

6.5 Editar el archivo `System.properties` que está en el directorio `%CATALINA_HOME%\salmonprops`, y especificar la ruta correcta de las siguientes propiedades:

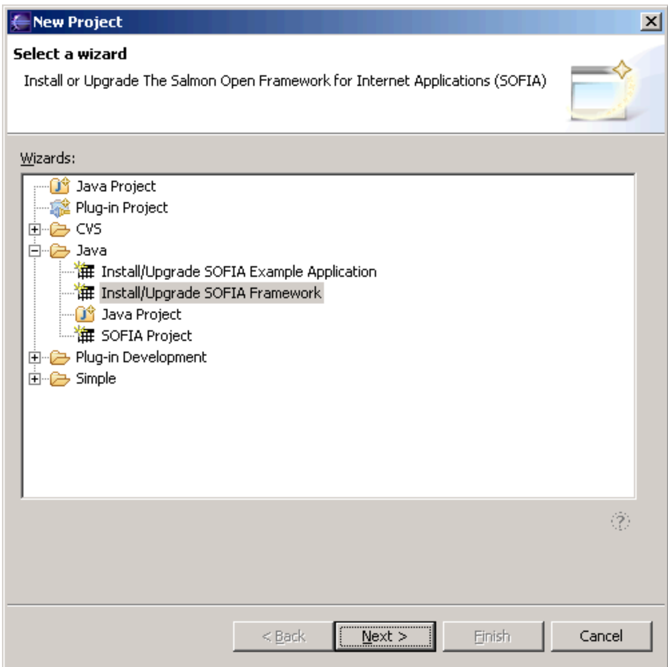
```
- TLDPath=%SOFIA_HOME%\\Resources\\Tomcat
- IDEFrameworkResourcesPath=%SOFIA_HOME%\\Resources
- IDEDreamweaverPath=%DREAMWEAVER_HOME%
- IDEBrowserPath=%BROWSER_DEFAULT_HOME%
- IDEBrowserPath.IE=% IEXPLORE_HOME%
- IDEBrowserPath.Netscape=%NETSCAPE_HOME%
```

Si no se tiene instalado estos navegadores, se puede omitir la definición de éstos (Internet Explorer y NetScape)

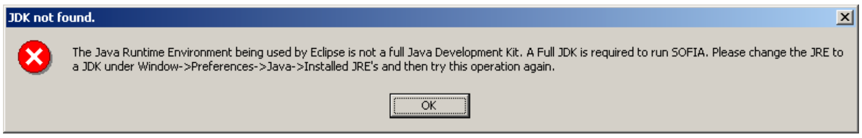
6.6 Para activar el menú y toolbar de SOFIA seleccionamos
Windows → Customize Perspective... y en la pestaña **Commands** de la ventana seleccionamos el grupo de comandos “**Salmon Menu**”



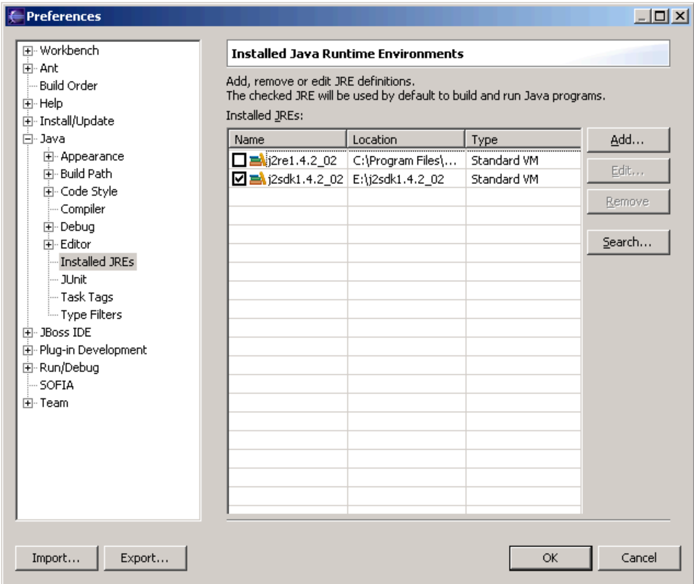
6.7 Seleccionamos File → New Project … y en el grupo de JAVA: **Install/Update SOFIA Frameworks** y presionamos el botón Next

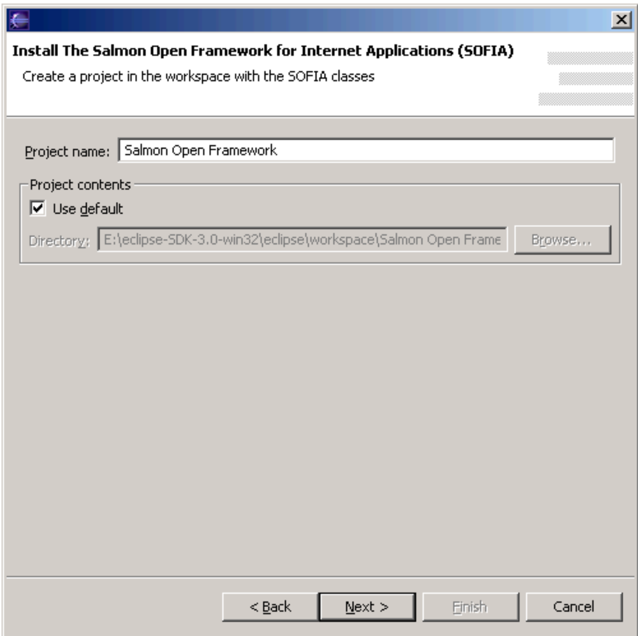


En el caso que se presente este mensaje de error, procedemos a corregir el JRE configurado para JAVA por el JSDK, caso contrario continuamos con la siguiente opción, 6.8.



Para corregir este mensaje de error, vamos a **Window → Preferences** y seleccionamos el grupo de Java: **Installed JREs.**
En esta opción adicionamos el J2SDK 1.4.

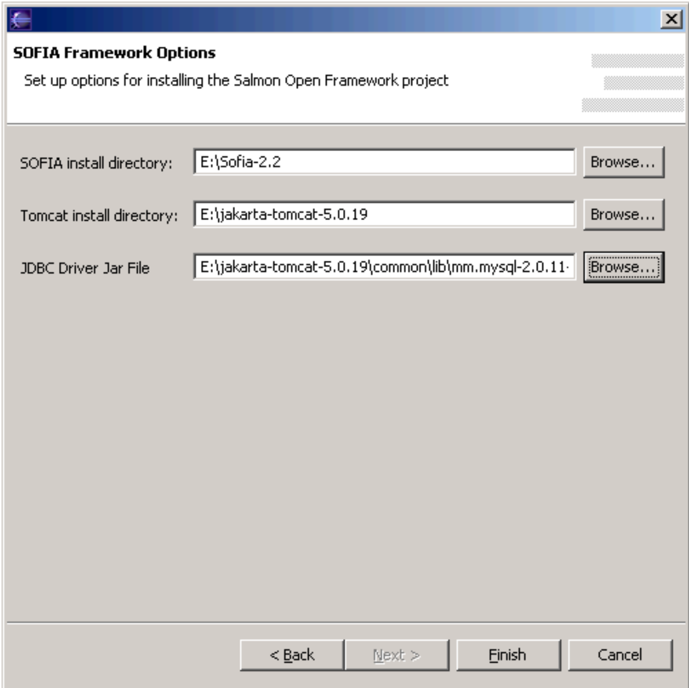




En esta ventana ingresamos las rutas:

```
- %SOFIA_HOME%,
- %CATALINA_HOME%
- %CATALINA_HOME%\common\lib\mm.mysql-2.0.14-bin.jar
```

Para la instalación de Sofia es necesario que el directorio %CATALINA_HOME% tenga dentro el directorio "salmonprops", caso contrario no se admitirá como directorio Tomcat válido.



6.8 Finalmente el proyecto se construirá y se observará sobre el Package Explore.

CONCLUSIONES

La instalación del Plugin de SOFIA versión 2-2 en la plataforma de Eclipse versión 3.0 no trabaja adecuadamente con la versión de TOMCAT 4.0 y 5.0. Está recomendado que para esta versión de SOFIA que se trabaje con la versión de Eclipse 2.0 o 2.1 y con la versión de TOMCAT 4.1. Por tal motivo no se puede levantar los servicios de TOMCAT desde Eclipse.

7. Instalación de JBoss

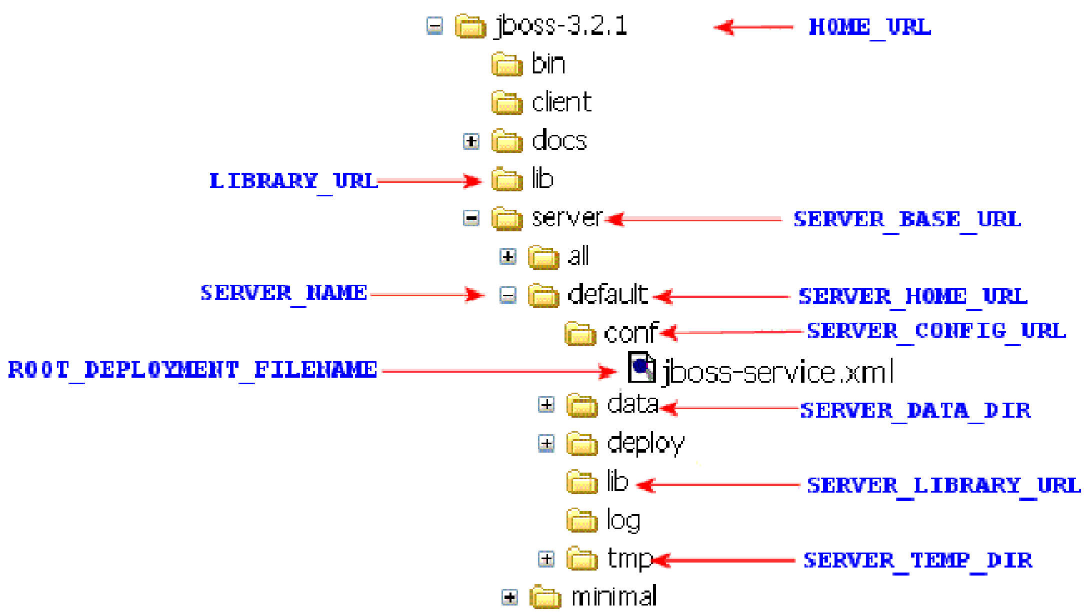
Instalación: Desempaquetar → `jboss-3.2.4.zip`
La versión de JBoss que estamos empleando es la 3.2.4

Revisiones realizadas:

7.1 Verificar que se encuentra instalado el JSDK versión 1.4 o superior.

```
C:\>java -version
java version "1.4.2_02"
Java(TM) 2 Runtime Environment, Standard Edition
(build 1.4.2_02-b03)
Java HotSpot(TM) Client VM (build 1.4.2_02-b03,
mixed mode)
```

7.2 La estructura del directorio es:



- bin : Contiene varios scripts y archivos asociados, como run.sh, shutdown.sh.
- client: Contiene configuración y archivos jar que puede necesitar el cliente o un contenedor externo.
- docs: Contiene los DTDs para los XML que se utilizan en JBoss, ejemplos de configuración de recursos y documentación de pruebas a JBoss.

- lib: Archivos jar que utiliza JBoss a bajo nivel.
- server: Cada subdirectorio aquí dentro especifica una configuración diferente para el servidor. La configuración que se escoja al momento de iniciar JBoss representará al servidor y todos los servicios mientras JBoss esté corriendo. Dentro de cualquiera de estos directorios habrá una estructura de directorios:
    - o conf: Contiene archivos de configuración, principalmente el *jboss-service.xml*.
    - o data: Almacena los datos de Hypersonic (embebido) y de la mensajería de JBoss JBossMQ.
    - o deploy: Los archivos war, jar o ear se ponen aquí para que el servidor los descomprima y los haga disponibles automáticamente (hot-deploy).
    - o lib: Librerías requeridas por el servidor.
    - o log: Donde se almacena información de monitores y errores. JBoss utiliza por defecto log4j.
    - o tmp: El servidor guarda temporalmente los archivos descompresos del directorio deploy.
    - o work: Utilizado por tomcat para compilar JSP.

Los directorios data, log, tmp y work son creados automáticamente por JBoss al momento de levantarse.

7.3 Hay tres formas de levantar JBoss:

`run –c all`	Todas las características del servidor
`run –c default`	Las características por defecto
`run –c minimal`	Lo mínimo que necesita JBoss para levantar el servicio

Para levantar los servicios de JBoss, vamos al directorio `%JBOSS_HOME%\bin`, y ejecutamos el archivo run.bat
La salida de la consola será la siguiente:

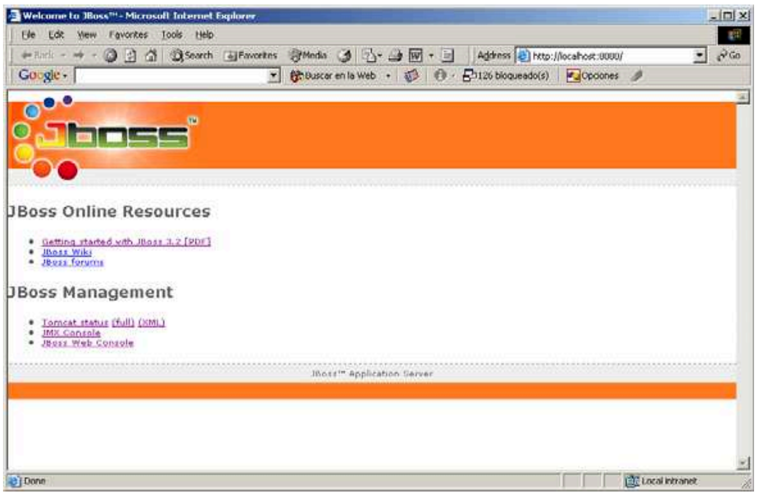
```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

E:\jboss-3.2.4\bin>run
===================================================
==========================
.
  JBoss Bootstrap Environment
.
```

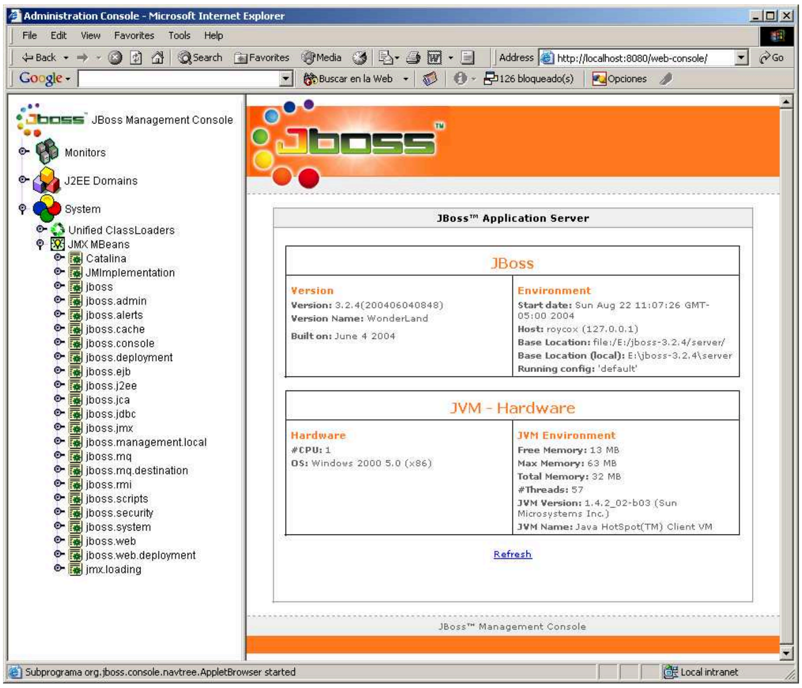
```
  JBOSS_HOME: E:\jboss-3.2.4\bin\\..
.
  JAVA: E:\j2sdk1.4.2_02\bin\java
.
  JAVA_OPTS:  -Dprogram.name=run.bat
.
  CLASSPATH:
E:\j2sdk1.4.2_02\lib\tools.jar;E:\jboss-
3.2.4\bin\\run.jar.
====================================================
.......
.......
11:08:14,671 INFO  [Http11Protocol] Arrancando
Coyote HTTP/1.1 en puerto http-0.0.0.0-8080
11:08:15,046 INFO  [ChannelSocket] JK2: ajp13
listening on /0.0.0.0:8009
11:08:15,078 INFO  [JkMain] Jk running ID=0
time=0/125  config=null

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

7.4 Verificación de la instalación a través de una conexión vía navegador a `localhost` por el puerto default. http://localhost:8080



7.5 En esta ventana podemos ingresar a los servicios de Administración de JBoss (JBoss Management Console), a través del link **Server Web Console.**



Servicios principales

Los servicios que primero se levantan están declarados en *conf/jboss-service.xml*. Aquí se podrán ver varios servicios como logging y JNDI.

El servicio de logging que utiliza JBoss el log4j. Existe el archivo de configuración *conf/log4j.xml*.
Log4j define varias salidas para los mensajes. Por defecto está la consola y los archivos dentro del directorio *log*. En los archivos se encuentra toda la información, mientras que por defecto en consola el nivel de registro de mensaje es INFO. Si se requiere más información se puede cambiar éste por DEBUG.

Más servicios pueden encontrarse en el directorio *deploy*. Pueden encontrarse en archivos xml o en archivos SAR de JBoss (Service ARchive), que contienen el xml declarando el servicio y recursos adicionales como clases y librerías.
Un servicio importante en el de Tomcat. El archivo *jbossweb-tomcat41.sar* contiene el servicio de contenedor web que viene embebido en JBoss.
Así mismo existe el servicio de base de datos Hypersonic (hsqldb-ds.xml), JavaMail (mail-service.xml), entre otros.

CONTENEDOR WEB

JBoss viene con Tomcat como contenedor web por defecto. El servicio de Tomcat está empaquetado como *jbossweb-tomcat41.sar* dentro del directorio *deploy*. Todos los archivos necesarios están aquí, al igual que un archivo web.xml de configuración básica para las aplicaciones web.
Tomcat se declara como un MBean dentro del archivo *META-INF/jboss-service.xml*.

Hay que tener en cuenta que las cosas son un poco distintas en el Tomcat embebido que cuando se utiliza Tomcat como un servidor stand-alone. Las aplicaciones web deben ser puestas dentro del directorio *deploy* de JBoss. JBoss está a cargo de los servicios embebidos y no debe ser necesario ingresar manualmente al directorio de Tomcat.
Los archivos log deben ser vistos en la consola y archivos log de JBoss.

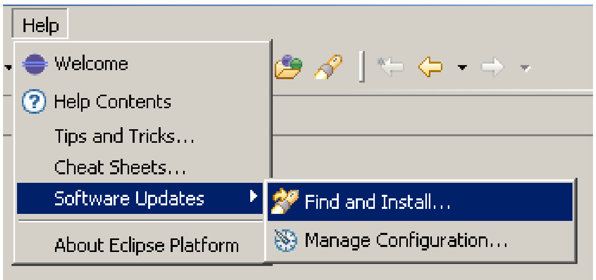
8. Instalación del Plug-in de JBoss para Eclipse

Instalación: Desempaquetar → `org.jboss.ide.eclipse_1.3.30.bin.dist.zip`
Revisiones realizadas:

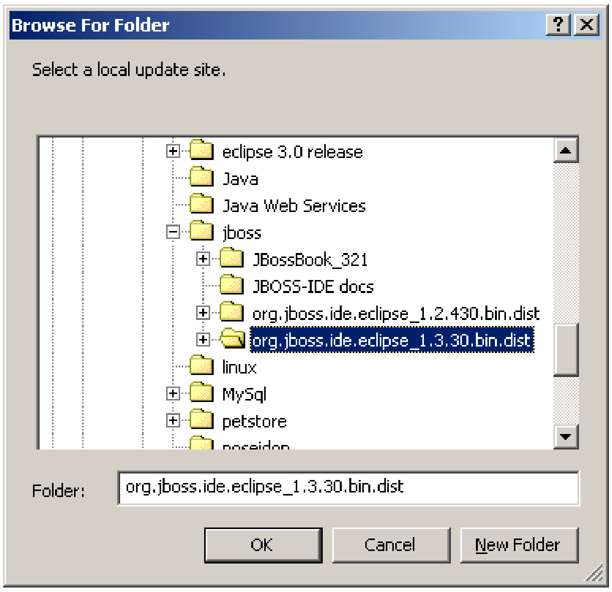
8.1 La plataforma empleada para esta instalación es Eclipse versión 3.0 con JBoss versión 3.2.4.

8.2 Iniciamos la aplicación de Eclipse y seleccionamos el menú:

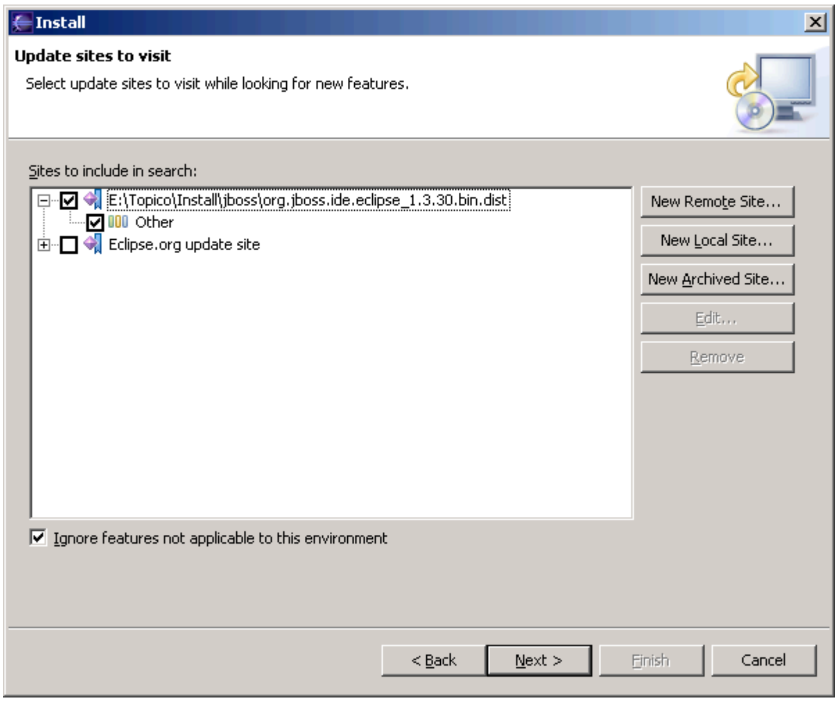
Help → Software Updates → Find and Install ...

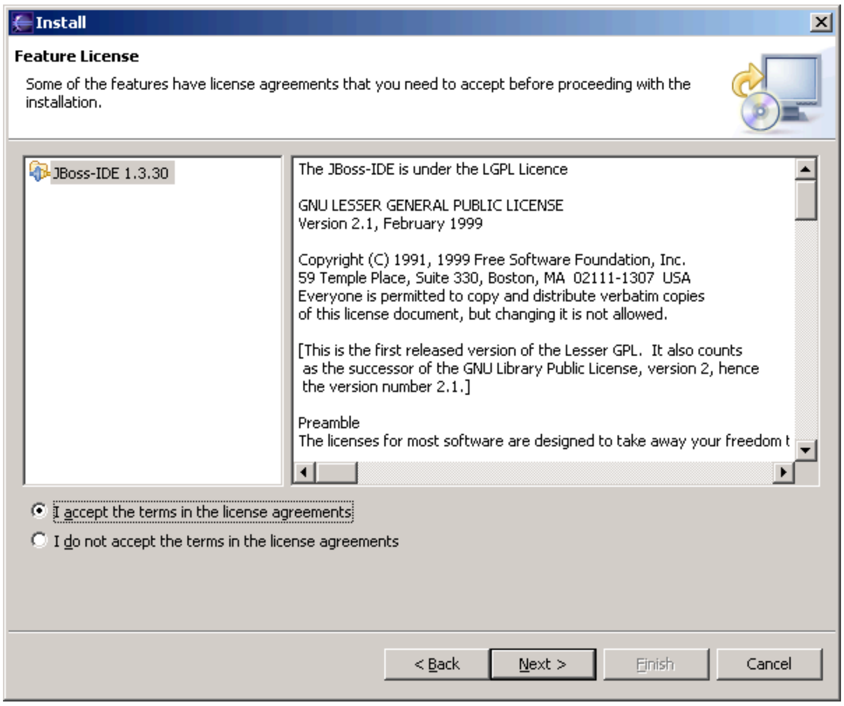


8.3 Seleccionamos la carpeta `org.jboss.ide.eclipse_1.3.30.bin.dist`

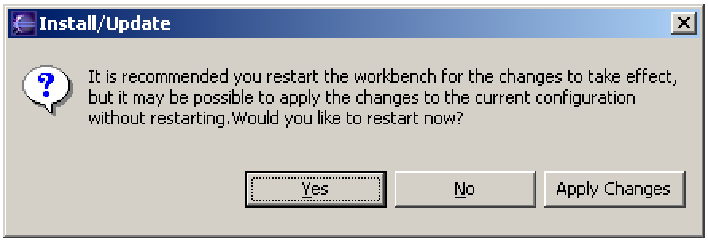


8.4 Seleccionamos el site a incluir **Other** y presionamos el botón **Next**



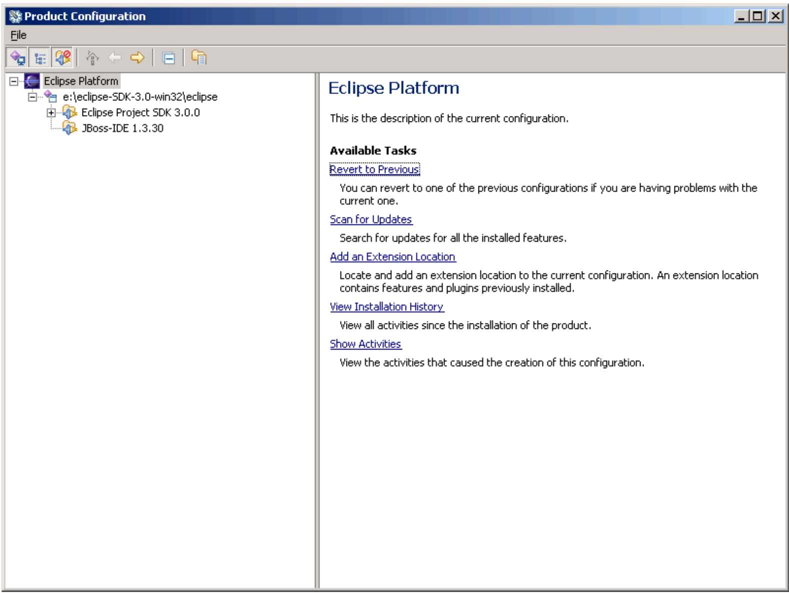


8.5 Una vez finalizado la instalación del plug-in se debe reiniciar Eclipse.



8.6 Para verificar la instalación correcta del plugin de JBoss podemos visualizar `Manager Configuration.`

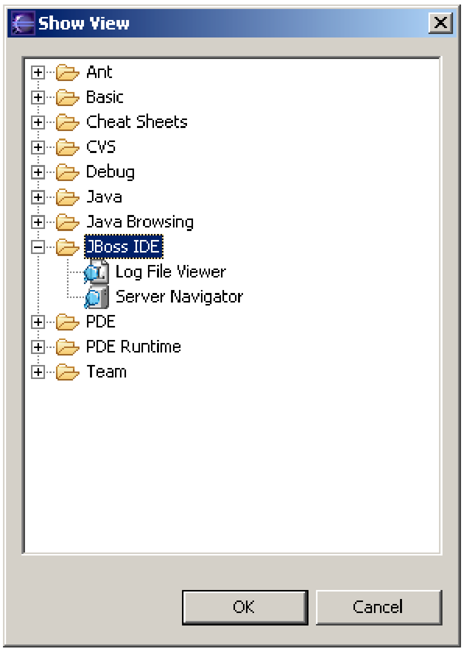
Help → Software Updates → Manager Configuration ...
Seleccionamos en la ventana la opción **View Installation History**



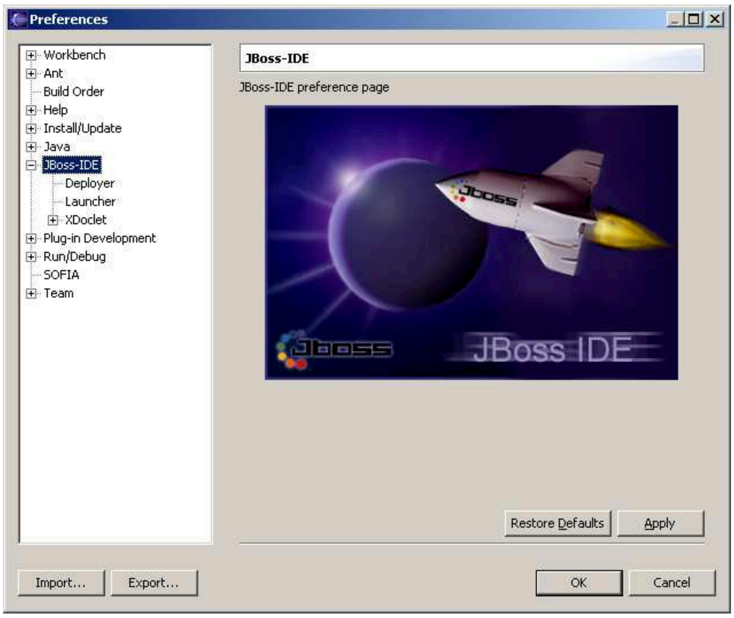
Aquí se puede observar la salida del estado de la instalación.

Sat Aug 21 15:59:35 GMT-05:00 2004			
Date / Time	**Target**	**Action**	**Status**
Sat Aug 21 15:59:21 GMT-05:00 2004	org.jboss.ide.eclipse_1.3.30	feature-install	success
Sat Aug 21 15:59:35 GMT-05:00 2004	org.jboss.ide.eclipse_1.3.30	feature-enable	success

Adicionalmente se puede seleccionar en el menú **Window → Show View → Other ...** y observar la ventana que contiene la vista previa para el JBoss IDE.



También lo podemos verificar con el menú **Window → Preferences ...**



ALCANCES – PARTE (1)

a. Plataforma**:**
- o Eclipse versión 3.0
- o JBoss versión 3.2.4 (Embebido Catalina - Apache Tomcat/5.0.26)

✓ Se actualizó el IDE de JBoss para Eclipse de la siguiente manera:
- o IDE Anterior: eclipse_1.2.430. (JBoss)
- o IDE Actual: eclipse_1.3.30. (JBoss)

✓ Se actualizó la versión del motor de base de datos MySQL
- o Versión Anterior: 3.23.54.
- o Versión Actual: 4.0

b. Se justifica **por qué** en la versión de Eclipse 3.0 el servicio de TOMCAT no se inicia a través del IDE de SOFIA versión 2.2 (Ver Conclusiones Temario 6 – Pág. 8)

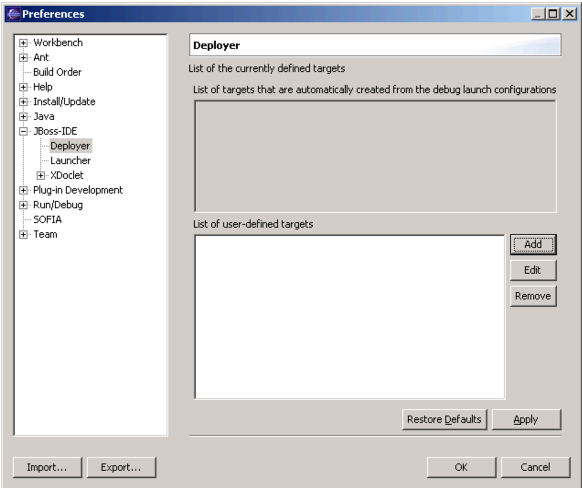
c. Se actualiza la documentación de estos cambios**.**

8. Levantar los Servicios de JBoss desde Eclipse

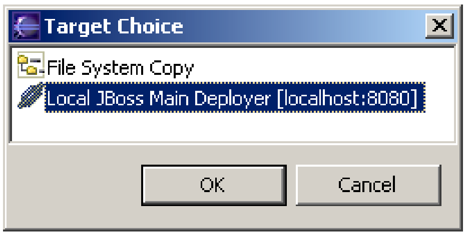
Plataforma

Eclipse versión 3.0
JBoss versión 3.2.4 (Embebido Catalina - Apache Tomcat/5.0.26)

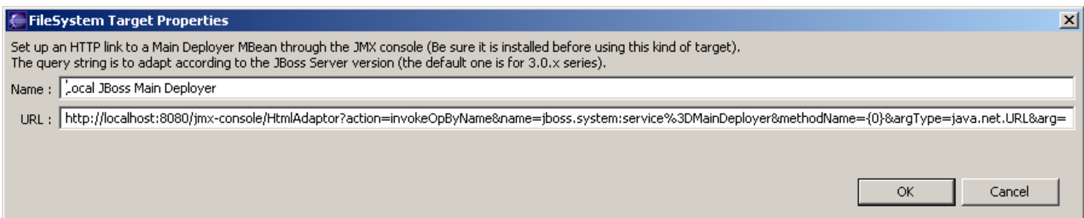
9.1 Seleccionamos el menú **Window → Preferences** y en el menú de **JBoss-IDE** seleccionamos **Deployer** (submenú)



9.2 Presionamos el botón **Add** y en la siguiente ventana OK.

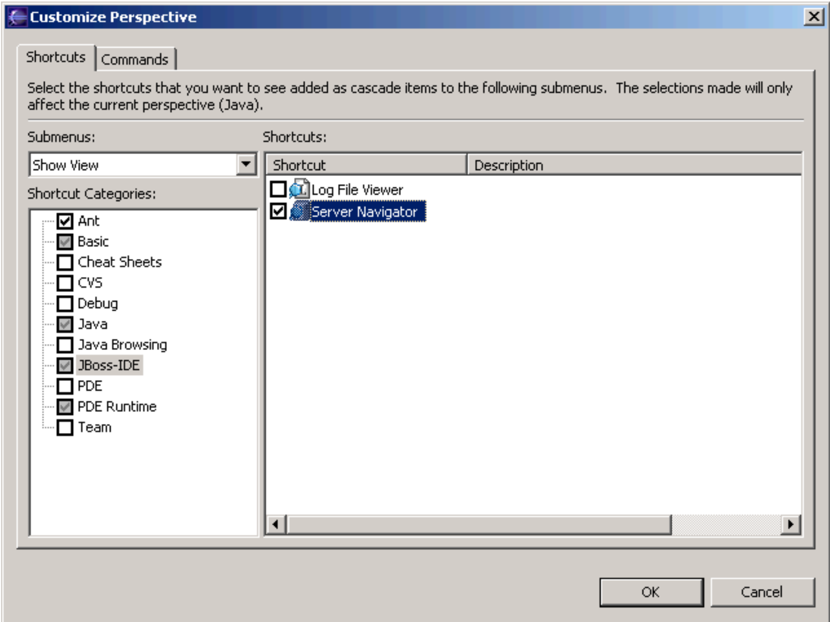


9.3 Luego presionamos el botón OK.

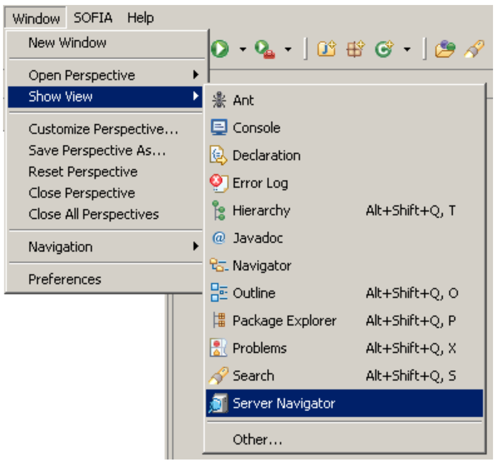


9.4 Finalizamos la ventana de **Preferences**, aceptando todos estos cambios realizados presionando OK.

9.5 Personalizamos las perspectivas del IDE de JBoss "Server Navigator" para el submenú **Show View**

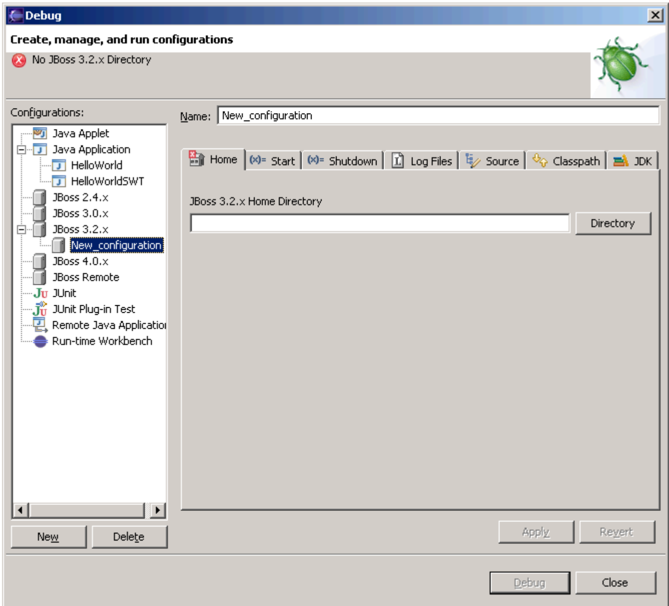


9.6 Seleccionamos la perspectiva:
Window → Show View → Server Navigator

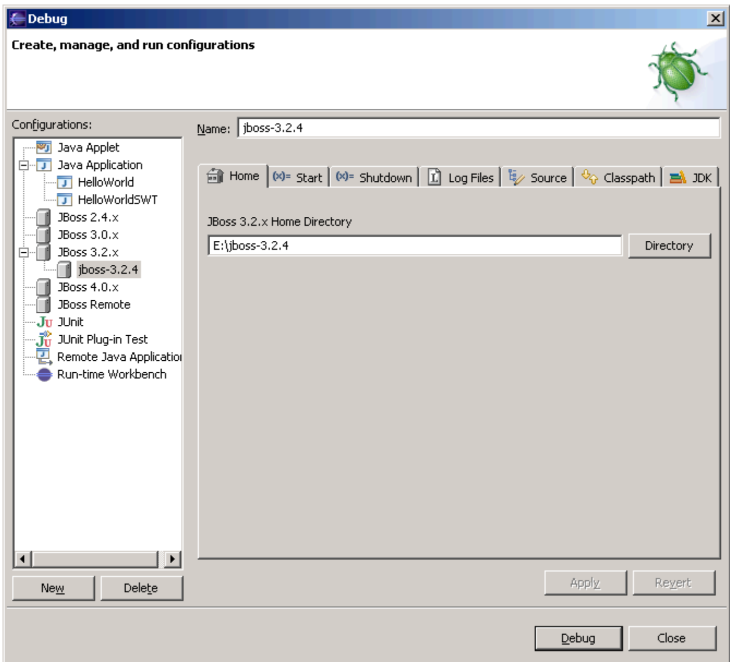


Seleccione el botón de configuración .

9.7 Se presentará la siguiente ventana, donde seleccionaremos **JBoss 3.2.x** en la sección de **Configurations.** Luego presionamos el botón **New**, donde se presentará un ítem **New_configuration**.



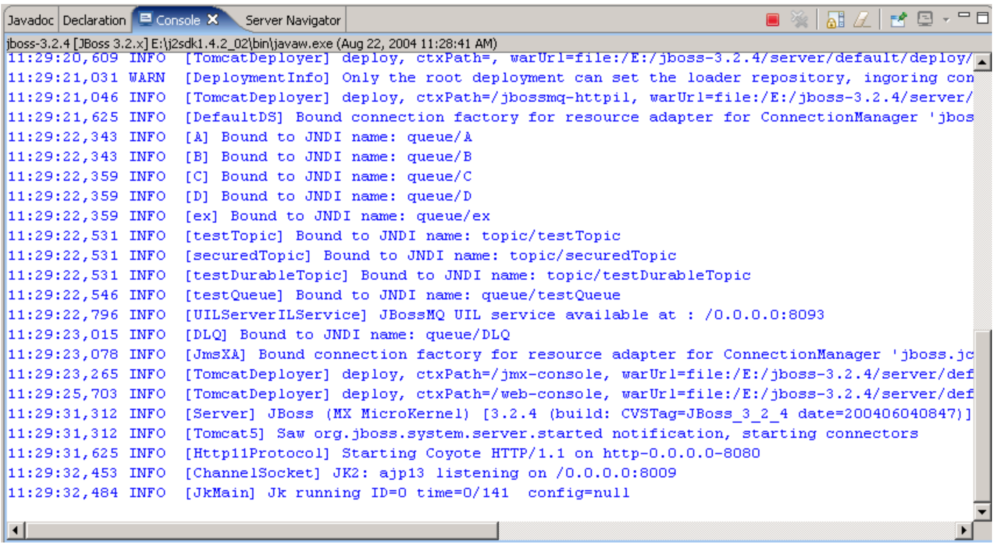
9.8 Presionamos el botón **Directory** para localizar el directorio `%JBOSS_HOME%`. Definimos el nombre del servidor "**jboss-3.2.4**" en la sección **Name** y presionamos el botón **Apply**



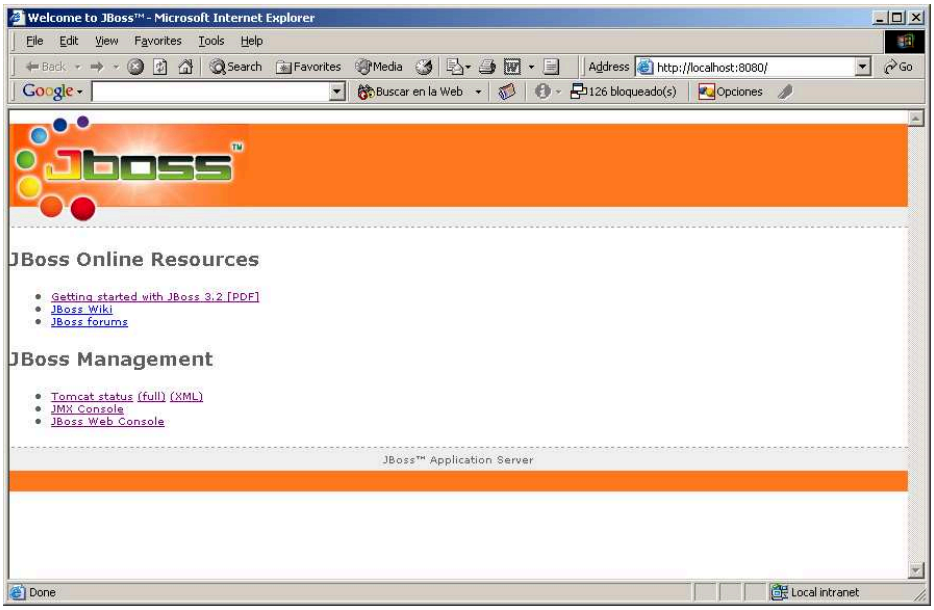
9.9 Cerramos la ventana y podemos observar en la parte inferior de la perspectiva de **Server Navigator** el nuevo servidor jboss3.2.4.

9.10 Presionamos el icono de **Start**  y se puede observar como el estado del servidor es **running**

Adicionalmente se puede escoger la pestaña **Console** para ver los servicios de JBoss como se ejecutan.



9.11 Para verificar que los servicios de JBoss se levantaron correctamente, realice una conexión vía browser a http://localhost:8080


Welcome to JBoss™ - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/
Go
Buscar en la Web
126 bloqueado(s)
Opciones
JBoss Online Resources
Getting started with JBoss 3.2 [PDF]
JBoss Wiki
JBoss forums
JBoss Management
Tomcat status (full) (XML)
JMX Console
JBoss Web Console
JBoss™ Application Server
Done
Local intranet

BIBLIOGRAFÍA

[1] Sun Microsystems, The Java EE 5 Tutorial, https://www.sun.com/offers/docs/JavaEETutorial.pdf, Septiembre 2007.

[2] JBoss Community, Installation and Getting Started Guide, www.jboss.org, 2008.

[3] Wikipedia, articulos varios, www.wikipedia.org, 2009.

[4] Apache Software Foundation, The Apache Tomcat 5.5 Servlet/JSP Container, http://tomcat.apache.org/tomcat-5.5-doc/index.html, 2009.

[5] Sun Microsystems, MySQL 5.1 Reference Manual, http://www.mysql.com/doc , 2009

[6] The Apache Sofware Foundation, Struts, http://struts.apache.org, 2008

[7] Hibernate Community, Hibernate Reference Documentation, http://www.hibernate.org/documentation, 2007

[8] David R. Heffelfinger, JasperReports for Java Developers, PACKT Publishing, 2006

[9] Werner Guttmann, Castor 1.3 – Reference documentation, The Castor Project, 2009

[10] Gary Cernosek - IBM, A Brief History of Eclipse, http://www.ibm.com/developerworks/rational/library/nov05/cernosek/ , 2009

[11] Eclipse Foundation, Eclipse 3.1 Documentation, http://www.eclipse.org/documentation/, 2009

[12] Trygve Reenskaug, MVC – Xerox PARC 1978 – 79, http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html, 2009

[13] Trygve Reenskaug, "The Model-View-Controller (MVC), Its Past and Present", University of Oslo, Agosto 2003

[14] Trygve Reenskaug, MODELS - VIEWS – CONTROLLERS (10-dic-1979), http://heim.ifi.uio.no/~trygver/, 2009