EXPERIMENTACIÓN Y EVALUACIÓN DEL FRAMEWORK JBOSS AOP

CESAR AUGUSTO HERRERA QUIROZ

IVÁN DARÍO PÁEZ ANAYA

DEPARTAMENTO DE SISTEMAS

UNIVERSIDAD EAFIT

MEDELLÍN

2007

EXPERIMENTACIÓN Y EVALUACIÓN DEL FRAMEWORK JBOSS AOP

CESAR AUGUSTO HERRERA QUIROZ

CODIGO: 200017504010

IVÁN DARÍO PÁEZ ANAYA

CODIGO: 200110084010

Trabajo del proyecto de grado presentado como requisito para aspirar al título de Ingeniero de Sistemas

ASESOR:

LENIN DAVID LOZANO

DEPARTAMENTO DE SISTEMAS

UNIVERSIDAD EAFIT

MEDELLÍN

2007

Nota de Aceptación

_____

_____

Presidente del Jurado

_____

Jurado

_____

Jurado

_____

_____

Ciudad, Fecha (día/mes/año)

A Dios por darme la vida y la oportunidad de realizar los deseos del corazón.
A mis padres Ernesto y Raquel, a mis hermanos, Marcela y Rolando,
por apoyarme y estar siempre conmigo.

Ivan Darío Paez Anaya

A mis padres Carlos Herrera y Ana María Quiroz, a mi hermana Sandra Herrera
y a mi Novia Paola Chica por estar siempre conmigo.

Cesar Augusto Herrera Quiroz

AGRADECIMIENTOS

Queremos expresar públicamente nuestros agradecimientos a:

Especialmente a la profesora Raquel Anaya, quien nos brindo constantemente su colaboración y asesoría durante todas las fases del proyecto. Haciendo posible la culminación exitosa de esta etapa de nuestras vidas.

A Lenin David Lozano, quien fue nuestro asesor de proyecto de grado, por sus comentarios y observaciones que siempre fueron muy pertinentes.

Al Grupo de Investigación de Ingeniería de Software y todos sus integrantes. Quienes nos permitieron encubar nuestras ideas, compartirlas y nos generaron al mismo tiempo nuevas inquietudes.

A la Universidad EAFIT y su plantel de profesores, que nos brindo la posibilidad de formarnos no solo como profesionales si no también como personas integrales para aportar buenas cosas a nuestra comunidad.

A nuestros amigos. Que compartieron tantas experiencias con nosotros y siempre nos comprendieron y apoyaron en todos los momentos.

TABLA DE CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

ANEXO A. GLOSARIO DE TÉRMINOS

Aspecto (Aspect): En una clase de java que encapsula un determinado numero de consejos (Advice), puntos de corte (pointcut) o clases mezcladas (mixins class), o cualquier otro constructor de JBoss AOP.

Anotación (Annotation): Este concepto aparece desde la versión Java 1.5.0. y sirve para poner marcas en el código que será utilizados por algún proceso de precompilación

Consejo (Advice): Este es el código adicional que queremos aplicar al sobre una funcionalidad existente.

Interceptor (Interceptor): Es un aspecto que solamente tiene un método consejo (Advice). El aspecto es una clase que implementa el servicio invoke declarado en la interfaz llamada Interceptor.

Introducción (Introduction): Es usado para forzar a determinada clase que ya existía a implementar una interfase o adicionar una anotación (Annotation) a alguna cosa.

Clase Mezclada (Mix-ins class): Esto sucede cuando un interceptor modifica una interfaz que esta siendo usada por cierta clase y le adiciona nuevos métodos o propiedades.

Punto de unión (Joinpoint): Representa una ejecución concreta de un evento donde el aspecto va a intervenir.

Punto de corte (Pointcut): Es un predicado que define los puntos de ejecución en flujo de la aplicación donde se desea agregar un comportamiento aspectual.

Problema cruzado (Cross-cutting concerns**):** Incluso aunque la mayoría de las clases de un modelo OO realizarán una función sencilla y específica, normalmente comparten requerimientos secundarios comunes con otras clases. Por ejemplo, podríamos querer

añadir logging a las clases dentro de la capa de acceso a los datos y también a las clases de la capa de presentación siempre que un thread entre o salga de un método. Incluso aunque la funcionalidad primaria de cada clase sea muy diferente, el código necesario para realizar la segunda funcionalidad es casi siempre idéntico

Framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje de scripting entre otros softwares para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

RESUMEN

En este proyecto de grado nos concentraremos a estudiar un framework de concepción aspectual como lo es JBoss AOP, el estudio de los frameworks es muy importante desde la perspectiva de la industria de software puesto que, si se desea que el enfoque de aspectos pueda ser utilizado en la solución de problemas reales, se hace necesario que los frameworks que facilitan la implementación de aspectos hayan alcanzado cierta madurez.

Este trabajo se enmarca del proyecto MMEDUSA que actualmente se esta desarrollando por el Grupo de Ingeniería de Software de la Universidad Eafit y una de las empresas de software de la ciudad, el cual tiene por objetivo definir un marco metodológico para la adopción de aspectos en la industria.

En el capitulo 2 de este documento se presenta el estado del arte de dos herramientas aspectuales. En el capitulo 3 se presenta un acercamiento mas detallado al framework de JBoss AOP integrado con JBoss AS. En el capitulo 4 y 5 se presenta el caso practico de estudio, implementado en el IDE de Eclipse-JBoss así como una descripción de las herramientas utilizadas, así como la implementación de cada uno de los aspectos realizados. En el capitulo 6 se presenta una evaluación del framework aspectual JBoss AOP de acuerdo a los criterios de calidad sistémica establecidos por el grupo de ingeniería de software de la universidad Eafit. Finalmente se presentan las conclusiones del trabajo realizado.

1. INTRODUCCIÓN

1.1 MOTIVACION

La ingeniería del software es una disciplina relativamente joven y con una acelerada evolución. En sus inicios, se tenía un código en el que no se tenía una estructura definida. Los datos y la funcionalidad se mezclaban sin una línea que los dividiera claramente. Los programas tenían una estructura de espagueti y el código era difícil de entender y mantener.

A medida que la complejidad del software fue aumentando, se vio la necesidad de establecer principios de ingeniería como la modularidad, la abstracción, el encapsulamiento, que determinaran la calidad de los programas. Esto dio origen a una programación de más alto nivel, con la noción de tipos de datos abstractos, bloques estructurados y agrupamientos de instrucciones a través de procedimientos y funciones, conocido como la programación estructurada. Más tarde la tecnología de orientación por objetos evoluciona el concepto de los tipos de datos abstractos, como módulos auto contenidos que representan tanto estructura como comportamiento, estos tipos establecen relaciones de generalidad, asociación y agregación, que permite representar la realidad de un espacio del problema. Los diferentes paradigmas de programación parten de la noción de descomponer un sistema complejo en subsistemas más fáciles de manejar, siguiendo la antigua técnica de “dividir y conquistar”.

Aunque la programación por objetos nos ayuda a descomponer el problema en unidades claramente identificables, dicha descomposición no logra representar la complejidad del problema en la realidad, ya que se hace necesario replicar funcionalidades referidas a un mismo asunto en cada una de las partes en las que fue

dividido inicialmente. Dicha situación ha sido denominada por Ossher and Tarr “La tiranía de la descomposición dominante” [1,2]

Consideremos una aplicación que incluya conceptos como seguridad, sincronización, concurrencia. Si bien estas funcionalidades pueden aparecer como módulos independientes, por ejemplo un componente de seguridad que permite la autorización y autenticación de usuarios, se requiere que los demás componentes accedan de manera consciente dicho componentes., lo cual implica que estos puntos de conexión se encuentran replicados en varias partes de la aplicación. El aporte de la orientación a aspectos es permitir la definición de los módulos que extienden funcionalidades ya existentes siguiendo el principio de inconsciencia. Es decir, el módulo que representa una funcionalidad transversal podrá intervenir una funcionalidad ya existente si que esta tenga que ser cambiada. El aspecto se convierte entonces en una especie de supervisor que esta encargado de conocer cuando y en donde tiene que hacer intervenir con el objetivo de extender o reemplazar una funcionalidad ya existente.

La orientación a aspectos surge entonces como un modelo de programación que ayuda a mejorar la modularidad de comportamientos comunes en más de un punto del sistema. Rápidamente este enfoque fue extendiéndose a otros frentes de trabajo en los cuales se busca potenciar el análisis de los asuntos de interés en todas las fases del proceso de desarrollo. Surgen inicialmente notaciones para especificar aspectos en UML que hacen abstracción de los constructores utilizados en AspecJ (pointcut, jointpoint, advice, etc.). Surgen luego trabajos para apropiar el enfoque aspectual en la fase de definición de requisitos, posteriormente en el análisis y diseño existen también trabajos que analizan el impacto de los aspectos en la arquitectura.

Otro de los frentes de trabajo que ha tenido gran actividad es la definición de soluciones de infraestructura que ayuden a implementar el manejo de aspectos. Buena

---

[1] H. Ossher, P. Tarr, Multi-dimensional separation of concerns using hyperspaces, IBM Research Report 21452, April, 1999.

[2] H. Ossher, P. Tarr, Using multidimensional separation of concerns to (re) shape evolving software, Communications of the ACM, October 2001, pp.43-50.

parte de estos frameworks surgen como evolución de soluciones objetuales existentes. El estudio de estos frameworks es muy importante desde la perspectiva de la industria de software puesto que, si se desea que el enfoque de aspectos pueda ser utilizado en la solución de problemas reales, se hace necesario que los frameworks que facilitan la implementación de aspectos hayan alcanzado cierta madurez. En este proyecto de grado nos concentraremos a estudiar un framework de concepción aspectual como lo es JBoss AOP.

Este trabajo se enmarca del proyecto MMEDUSA que actualmente se esta desarrollando por el Grupo de Ingeniería de Software de la Universidad Eafit y una de las empresas de software de la ciudad, el cual tiene por objetivo definir un marco metodológico para la adopción de aspectos en la industria.

En el capitulo 2 de este documento se presenta el estado del arte de dos herramientas aspectuales. En el capitulo 3 se presenta un acercamiento mas detallado al framework de JBoss AOP integrado con JBoss AS. En el capitulo 4 y 5 se presenta el caso practico de estudio, implementado en el IDE de Eclipse-JBoss así como una descripción de las herramientas utilizadas, así como la implementación de cada uno de los aspectos realizados. En el capitulo 6 se presenta una evaluación del framework aspectual JBoss AOP de acuerdo a los criterios de calidad sistémica establecidos por el grupo de ingeniería de software de la universidad Eafit. Finalmente se presentan las conclusiones del trabajo realizado.

1.2 OBJETIVOS

1.2.1 Objetivo general

Conocer las características del framework JBoss AOP de acuerdo a los criterios definidos en el proyecto MMEDUSA y analizar que tan viable es la utilización de una herramienta de este tipo en el desarrollo de proyectos reales.

1.2.2 Objetivos específicos

- Realizar un acercamiento teórico de las características de los frameworks aspectuales existentes.
- Estudiar las características del framework JBoss AOP, integrado con JBoss AS.
- Desarrollar un caso práctico que sirva como referencia para la industria y la academia en el uso del framework JBoss AOP integrado con JBoss AS.
- Realizar una evaluación del framework JBoss AOP de acuerdo a los criterios preestablecidos por el Grupo de investigación de ingeniería de sistemas de la Universidad Eafit.

1.3 ALCANCE DEL PROYECTO DE GRADO

Se partirá de los criterios de comparación de frameworks aspectuales determinados en le proyecto MMEDUSA. Este proyecto abarca el estudio de los conceptos del desarrollo de software orientado aspectos en el campo de las soluciones de frameworks, el estudio del framework JBoss AOP, la instalación del producto en el laboratorio de Ingeniería de Software, el modelado del caso práctico utilizando uno de los enfoques de diseño aspectual, la implementación del caso en la herramienta seleccionada y la evaluación de herramienta utilizando los criterios ya definidos.

Los productos resultantes son:

- La descripción del frameworks JBoss AOP.
- Caso de aplicación práctico, donde se demuestra la funcionalidad y viabilidad de la implementación de esta herramienta en la industria.
- Una evaluación del framework JBoss AOP con respecto a los criterios preestablecidos.
- Presentación de resultados al grupo de ingeniería de software.

2. ESTADO DEL ARTE FRAMEWORKS ASPECTUALES

El referente de partida más importante en este tema es el estudio realizado por el grupo AOSD EUROPE sobre las propuestas de middleware aspectuales.[2] Este fue un estudio que realizó la comunidad europea de aspectos y donde propone las características deseables que debe tener un framework aspectual para que pueda ser considerado útil para la implementación de aplicaciones orientadas a aspectos.

Los criterios de comparación aplicados sobre los frameworks aspectuales que propone AOSD EUROPE son los siguientes.

Modelo de programación orientada a aspectos. Este criterio de comparación es importante ya que los modelos de programación de los frameworks orientados a aspectos que estamos estudiando, pueden ser una extensión de los modelos orientados a objetos o de los modelos basados en componentes que existen actualmente, tales como CORBA, CMM/CORBA, EJB/J2EE. Por otra parte los frameworks aspectuales pueden tener su propio modelo de programación.

Entidades primarias soportadas. Este criterio hace referencia los frameworks aspectuales que puede soportar la separación de aspectos, en aplicaciones orientadas a objetos, basadas en componentes, orientadas por agentes, etc. De tal forma que se pueden trabajar integrados los nuevos conceptos con los anteriores. Es importante que los desarrolladores de software conozcan esta información antes de definir que framework aspectual van a utilizar.

Modelo de tejido estático versus dinámico. Partiendo que los aspectos deben estar tejidos con las clases de la aplicación. Este criterio hace referencia al tipo de tejido que pueden tomar. Por una parte este tejido se puede dar en tiempo de compilación, a este tejido se le llama tejido estático. O también el tejido se puede dar en tiempo de

[2] AOSD-EUROPE, 2005. Survey of Aspect-oriented middleware.

despliegue y/o de ejecución, a este es el modelo de tejido se le denomina, tejido dinámico.

Modelo de puntos de unión, invasivo versus no invasivo. Este criterio hace referencia al punto donde intervienen los aspectos para modificar el comportamiento de una entidad, ya sea una clase, un método, una propiedad, etc. Si los aspectos intervienen directamente una entidad llámese clase, método o propiedad para modificar su comportamiento, se dice que es un modelo de tejido invasivo. Por el contrario si los aspectos solamente pueden intervenir estas entidades a través de clases del tipo de interfaces públicas, a esto se denomina un modelo de tejido no invasivo.

Reusabilidad de aspectos. La reutilización de los aspectos es una de los principales atractivos de la programación orientada a aspectos. Ya que permitiría a los desarrolladores reducir el tiempo y los costos de intervenir una aplicación cuando esta ya esta terminada. La inclusión de los puntos de corte del aspecto junto con los consejos del aspecto evita que los aspectos sean reutilizados en diversos contextos, por esta razón se busca que los puntos de unión estén definidos aparte de la funcionalidad del aspectos, para permitir su reutilización independiente del contexto.

Aplicación de extensibilidad/adaptabilidad. Las entidades de software que son partes del entregable final de la aplicación, deben poder ser extendidas o adaptadas, en el diseño o en tiempo de ejecución. El tejido dinámico soporta adicionar, eliminar o actualizar las entidades de la aplicación en tiempo de ejecución. Por otra parte la definición de los puntos de unión en un lenguaje declarativo, en vez de programarlos, esto hace más fácil adaptar las relaciones de los aspectos a través de las entidades principales de la aplicación en tiempo de diseño y de ejecución.

Estos criterios fueron tomados como un punto de partida para el proyecto y fueron extendidos con criterios enfocados a la práctica del desarrollo de software en la realidad.[3] Este estudio clasifico los criterios de comparación de frameworks en cuatro

[3] Santiago Henao, Alexander Cañas, David Lozano. Aspect Oriented Middleware Comparison. Reporte técnico grupo de Ingeniería de softwafre, Universidad EAFIT. 2006.

categorías, criterios aspectuales, criterios de desarrollo, criterios generales, criterios de calidad y servicio.

Criterios aspectuales. La meta de esta categoría se evaluar como es el procedimiento de implementación de los conceptos de la programación orientada a aspectos. Tales conceptos como modelos de punto de unión, modelo de tejido, se convierten en el objeto de estudio de esta categoría.

Criterios de desarrollo. Esta categoría reúne los criterios que están directamente relacionados con el proceso de desarrollo de software. Además de aquellos que tienen relación con la plataforma que soporta el framework aspectual, podemos encontrar criterios como soporte de la herramienta y sintaxis de los aspectos.

Criterios generales. Esta categoría reúne a los criterios que están enfocados a determinar que tan viable puede ser el desarrollo de aplicaciones de software en los frameworks aspectuales, entre los criterios de desarrollo podemos encontrar, documentación del framework disponible, tipo de licenciamiento, la madurez del framework entre otros.

Criterios de calidad y servicio. Los criterios de calidad y servicio están direccionados a los requisitos no funcionales del software.

Antes de hablar del Framework JBoss AOP, vamos a dar un repaso por las otras alternativas en cuanto a frameworks aspectuales. A continuación vamos a hablar acerca de algunos framework aspectuales y que características tiene cada uno de ellos.

Los análisis que se muestran a continuación es el resultado de una recopilación de las investigaciones realizadas por integrantes del Grupo de Investigación en Ingeniería de Software de la Universidad Eafit. Este análisis fue esencial para complementar nuestro estudio del estado del arte de los framework aspectuales. Este acercamiento nos permitió hacernos una visión integrada de lo que se encuentra en el mercado de los frameworks existentes y las características de cada uno de ellos.

2.1 ESTUDIO DEL FRAMEWORK JAsCo [López, 2007]

2.1.1 Presentación de la Herramienta

JAsCo es un lenguaje de programación orientado a aspectos (AOP o AOSD) originalmente creado para el campo basado en componentes. Las principales contribuciones del lenguaje JAsCo son sus aspectos altamente reusables, que no necesitan ser escritos para un contexto concreto donde el advice tiene que aplicar, y su fuerte mecanismo de composición aspectual para manejar combinaciones de aspectos.

JAsCo permite hacer polimorfismo de aspectos por medio de herencias y de otros mecanismos como los métodos refinables dentro de los hooks.

JAsCo soporta un amplio rango de conceptos AOP: poincuts similares a los de AspectJ, aspectos independientes, despliegue explícito en conectores, combinaciones expresivas de aspectos, aspectos con estados, mezclas virtuales y programación adaptativa. Un inconveniente de todo esto es que el aprendizaje de JAsCo es más difícil que el de otros lenguajes. Más aún, se recomienda que las personas que estén empezando con la programación orientada a aspectos comiencen experimentando con AspectJ o AspectWerkz para aprender las bases.

Algunas otras características del lenguaje son:

- Despliegue avanzado de aspectos:
    - Disparo de eventos de Java Beans.
    - Metadatos en los poincuts con el uso de anotaciones de Java.
    - Aspectos de Aspectos.
- Composición avanzada de aspectos:
    - Estrategias de precedencia explícitas basadas en instancia de un conector. Como tal, los aspectos no tienen que escribir relaciones de

precedencia y la precedencia de varios aspectos combinados puede cambiar en otras aplicaciones diferentes de esos aspectos.

    - Estrategias de combinación explícitas en un conector que pueden manejar el comportamiento combinado de un conjunto de aspectos combinados.

- Programación adaptativa usando conectores transversales.
- Aspectos con estado (stateful aspects), por ejemplo, aspectos que especifican poincuts basados en protocolo.

La tecnología JAsCo se especializa en proveer integración y remoción dinámica de aspectos con un mínimo overhead, para habilitar o deshabilitar un aspecto, simplemente basta con mover un archivo y el cambio se ve reflejado inmediatamente en tiempo de ejecución. Además de las características semánticas que agrega a Java, JAsCo provee herramientas que permiten hilado de aspectos en tiempo de ejecución.

2.1.2 Arquitectura de la herramienta

JAsCo es una extensión orientada a aspectos de Java, que se queda tan cerca como le es posible a la sintaxis y conceptos originales de Java. Introduce dos nuevos conceptos: Aspect Beans y Connectors.

Un Aspect Bean es altamente reusable porque permite la descripción de intereses cruzados independientemente de los tipos de los componentes concretos y APIs, solo se enfoca en describir el “qué”, es decir los advice.

Un Connector por otro lado, despliega uno o más Aspect Beans dentro del contexto de un componente concreto y permite especificar una combinación de sus comportamientos aspectuales. Como su nombre lo indica, permite conectar aspectos con la aplicación inclusive en tiempo de ejecución. Dentro de él se especifican los

poincuts. También permite dar orden a la ejecución de advice cuando tienen un mismo poincut.

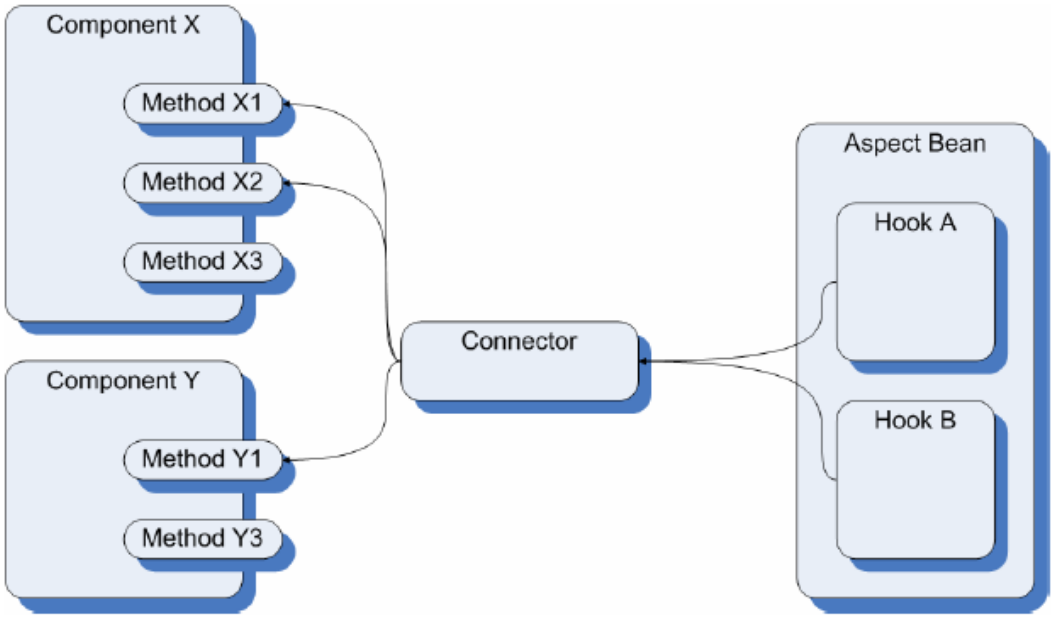

Figura 1: Arquitectura de JAsCo en tiempo de compilación.

En tiempo de Ejecución.

La tecnología JAsCo está basada en un tejido genuino en tiempo de ejecución que permite agregar, alterar y remover aspectos mientras la aplicación está siendo ejecutada.

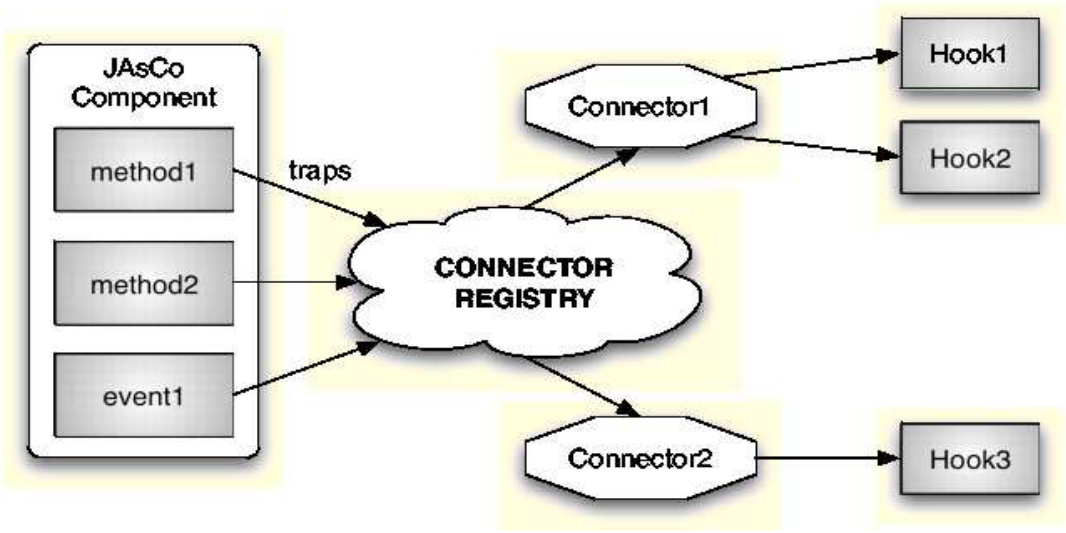

Figura 2: Arquitectura de JAsCo en tiempo de ejecución.

JAsCo utiliza un compilador en tiempo de ejecución llamado Jutta, con el que compila, optimiza y hace caché del código definido en los hooks.

También, utiliza un sistema de intercepción conocido como HotSwap que le permite hacer manipulaciones del bytecode para insertar el código compilado de los aspectos dentro de los métodos intervenidos. Estas inserciones sólo se llevan a cabo cuando es necesario, para evitar realizar operaciones redundantes, igualmente, realiza caché de esos cambios para ejecuciones futuras más rápidas.

En JAsCo también existe un tejedor en tiempo de compilación, que está compuesto por herramientas de transformación de AspectBeans. Éste no es excluyente con el HotSwap, y ambas herramientas se pueden combinar para obtener el máximo desempeño, por ejemplo, si se sabe que un aspecto siempre actúa sobre determinadas clases, se pueden preprocesar en tiempo de compilación, para reducir el Overhead causado en tiempo de ejecución por el HotSwap.

2.2 ESTUDIO DEL FRAMEWORK SPRING [Hincapié, 2007]

2.2.1 Presentación de la herramienta

Spring es un framework que provee una solución liviana para construir aplicaciones empresariales, que a su vez soporta la posibilidad de usar administración de transacciones, acceso remoto a la lógica usando RMI o servicios Web y varias opciones de persistir los datos en una base de datos. Spring provee un framework MVC con todas las características necesarias y formas transparentes de integrar la programación orientada por aspectos (AOP) en el software.

Spring es modular, lo que permite usar sólo las partes de él que son necesarias, sin tener que involucrar el resto. Spring ha sido diseñado para ser no invasivo, lo cual significa que las dependencias dentro del framework mismo son nulas (o mínimas dependiendo del área de uso).

Uno de los componentes de Spring está relacionado con AOP y permite introducir en el framework una manera más simple y poderosa de escribir aspectos personalizados, ya sea usando una aproximación basada en esquemas ó el estilo de anotación de AspectJ. Ambos estilos permiten el uso de advice de varios tipos y el uso del lenguaje de pointcuts de AspectJ, mientras que usan Spring AOP para hacer el tejido.

2.2.2 Arquitectura de la herramienta

El framework Spring contiene una gran cantidad de características las cuales están organizadas en siete módulos que se muestran en la figura 3.

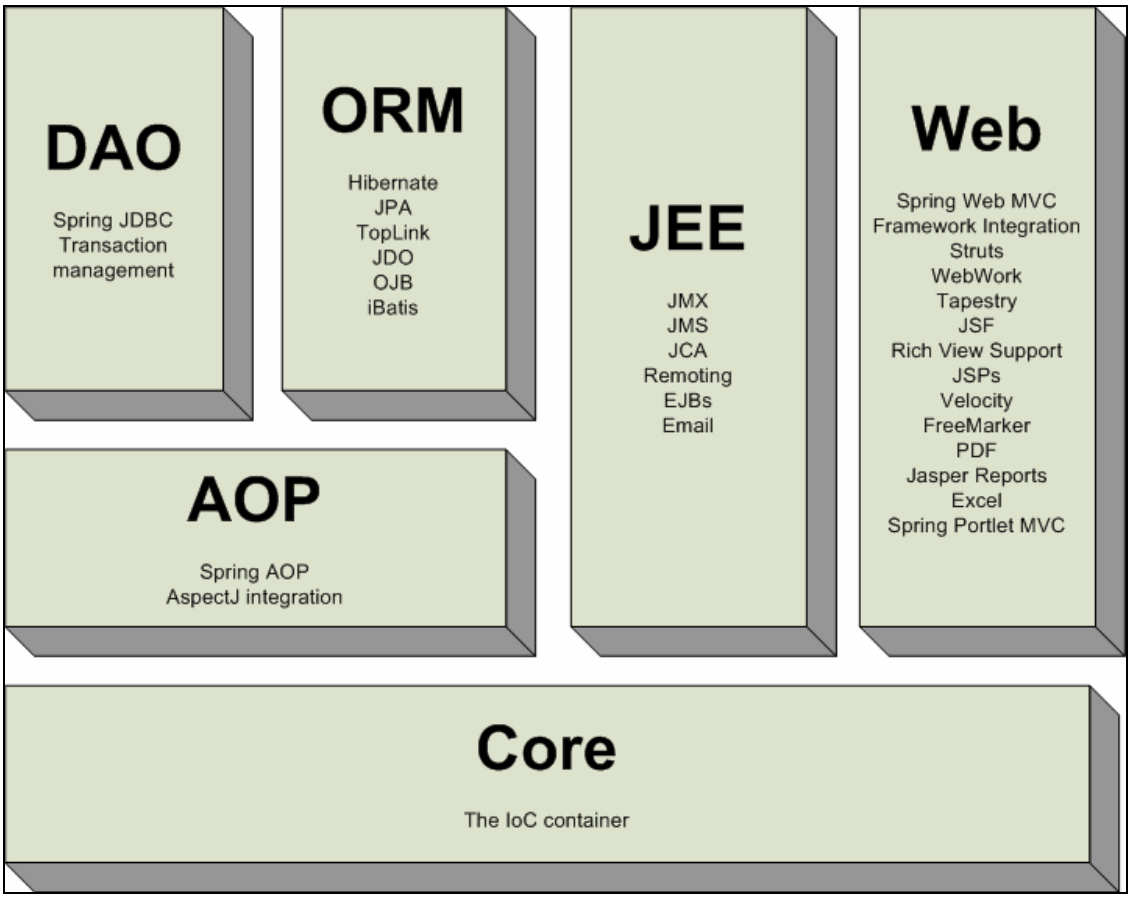

Figura 3: Módulos del Framework Spring

El módulo Core es la parte mas fundamental del framework y provee el contenedor IoC (Inversion of Control) y características de inyección de dependencias y sirve como base para el resto de los componentes. El concepto básico en este módulo es el BeanFactory el cual provee una implementación sofisticada del patrón factory, lo cual elimina la necesidad de singletons y permite el desacoplamiento de la configuración y la especificación de dependencias de la lógica real del programa.

El módulo Context provee mecanismos de acceso a objetos siguiendo el estilo de un framework que de alguna manera evoca a un registro JNDI. Este componente adiciona soporte para la internacionalización (I18N), propagación de eventos, carga de recursos y la creación transparente de contexto por medio de, por ejemplo, un contenedor de servlets.

El módulo DAO provee una capa de abstracción JDBC que remueve la necesidad de la codificación y procesamiento JDBC de los códigos de errores específicos de los motores de bases de datos. También, este módulo permite hacer administración de transacciones de una forma tanto programática como declarativa, no sólo para clases que implementen interfaces especiales, sino para todos los POJOs (Plain Old Java Objects).

El módulo ORM provee capas de integración para APIs populares para el mapeo objeto relacional, tales como JPA, JDO, Hibernate e iBatis. Con el módulo ORM se pueden usar todos los mapeadores en combinación con todas las demás características que Spring ofrece.

El módulo AOP provee una implementación de la programación orientada por aspectos que permite definir, por ejemplo, interceptores de métodos y pointcuts para desacoplar el código que implementa funcionalidades que, lógicamente hablando, deben estar separadas. Usando funcionalidades a nivel de meta datos también se puede incorporar en el código todo tipo de información sobre el comportamiento, en una manera similar a los atributos en .NET.

El módulo WEB de Spring provee características básicas de integración orientadas a la Web, tales como funcionalidades de carga de archivos en varias partes, la inicialización del contenedor IoC usando servlets escuchadores y contextos de aplicaciones orientadas a la Web. Cuando se usa Spring junto con WebWork o Struts, este es el módulo con el cual se hace la integración. Además de esto, este módulo también provee una implementación del patrón MVC (Model-View-Controller) para aplicaciones Web, el cual presenta una separación limpia entre el código del modelo del dominio y los formularios Web, permitiendo el uso de todas las demás características de Spring.

3. ESTUDIO DEL FRAMEWORK JBOSS AOP INTEGRADO CON JBOSS AS

En este capítulo se realiza un estudio mas detallado del framework JBoss AOP integrado con JBoss AS, que es el servidor de aplicaciones de JBoss objeto de estudio de este proyecto. Empezaremos hablando de la manera como se encuentra conformado en su interior el servidor de aplicaciones de JBoss. También vamos a explicar todos los servicios que provee y como se integra con JBoss AOP.

3.1 ESTUDIO DE JBOSS APLICATION SERVER

3.1.1 Servidor de aplicaciones

Un servidor de aplicaciones es un procesador en una red de computadores que ejecuta ciertas aplicaciones de software, proporcionando servicios de aplicación a las computadoras cliente. Gestiona la mayor parte de las funciones de lógica de negocio y de acceso a los datos de la aplicación. Un ejemplo común son los portales de Internet, que ofrece a las empresas la posibilidad de gestionar y divulgar su información de manera segura.

Tipos de servidores de aplicaciones

Los servidores de aplicaciones se encuentran disponibles sobre una gran variedad de plataformas, como Unix, Windows y GNU/Linux. Debido a la amplia difusión del lenguaje Java, el término servidor de aplicaciones se tiende a asociar directamente a la plataforma J2EE. Para propósitos del presente trabajo clasificaremos entre

servidores de aplicaciones J2EE y no J2EE. J2EE define una especificación de referencia para aplicaciones distribuidas que utilizan el lenguaje Java.

Servidores de aplicaciones J2EE

WebSphere (IBM), Oracle Application Server (Oracle Corporation) y WebLogic (BEA) están entre los servidores de aplicación J2EE comerciales más conocidos. EAServer (Sybase Inc.) es también conocido por ofrecer soporte a otros lenguajes diferentes a Java, como PowerBuilder. El servidor de aplicaciones JOnAS, desarrollado por el consorcio ObjectWeb, fue el primer servidor de aplicaciones libre en lograr certificación oficial de compatibilidad con J2EE. JBoss es otro servidor de aplicaciones libre y muy popular en la actualidad [4]

Servidores de aplicaciones NO J2EE

Con el aumento de la popularidad de .NET, Microsoft califica a su producto Internet Information Server como un servidor de aplicaciones. Adicionalmente, se pueden encontrar servidores de aplicación de código abierto y comercial de otros proveedores; algunos ejemplos son Base4 Server y Zope.

Servidor de Aplicaciones JBoss

JBoss es un servidor de aplicaciones de código abierto construido completamente en el lenguaje Java, JBoss implementa todo el paquete de servicios de J2EE ofreciendo una plataforma de alto rendimiento para aplicaciones de e-business. El propósito de JBoss es proveer una tecnología de capa media a bajo costo y de fuente abierta, y soportada en por servicios tecnológicos provistos por expertos.

JBoss actualmente cumple con los certificados de J2EE. Posee también una arquitectura flexible que lo convierten en una buena elección si se esta comenzando

[4] http://es.wikipedia.org/wiki/Servidor_de_aplicaciones

con aplicaciones J2EE. Se puede obtener fácilmente el código fuente y los binarios desde el sitio del proyecto en sourceforge [5]

El servidor de JBoss esta dividido en módulos y esta implementado usando plug-ins basados en componente Esto es posible gracias a que su microkernel esta basado en JMX, que es un API de Java, *Java Management Extension,* JMX.

Características

- Producto con licencia de código abierto: esto significa que JBoss AS puede ser descargado, instalado, utilizado y distribuido sin restricciones por la licencia.

- Cumple con los estándares J2EE: JBoss AS esta implementado en Java puro. Como una de las principales características al estar basado en Java, JBoss AS puede ser utilizado en cualquier sistema operativo que lo soporte. Este cumplimiento de los estándares lo convierten en un framework capaz de soportar aplicaciones empresariales.

- Confiable a nivel de empresa: Los principales desarrolladores de JBoss AS trabajan para una empresa de servicios, JBoss Inc., adquirida por Red Hat en Abril del 2006. Este proyecto está apoyado por una red mundial de colaboradores lo que inspira confianza a las empresas que lo utilizan.

- Arquitectura orientada a servicios: esto le permite a las empresas que implementen soluciones de software utilizando JBoss AS, configurar los servicios que provee el servidor y que mejor se adapten a sus necesidades empresariales.

- Servicios de middleware (software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas) para cualquier objeto de Java.

[5] http://sourceforge.net/projects/jboss

3.1.2 Arquitectura de la herramienta JBoss AS

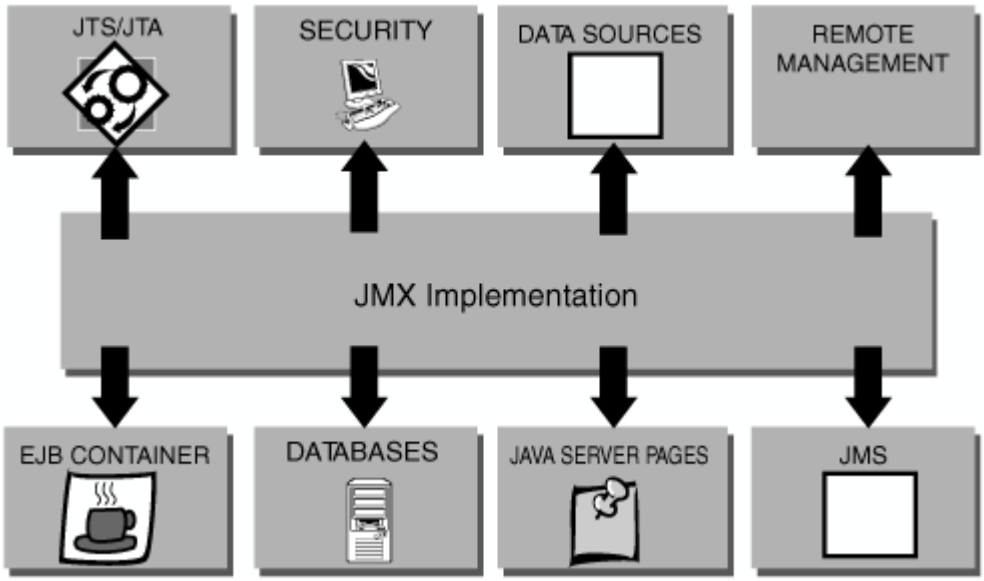

Figura 4: Integración de los módulos de JBoss AS usando JXM [6]

El éxito de las aplicaciones de código abierto de J2EE es que usen JXM. JXM es una buena herramienta para la integración de software ya que provee una columna vertebral que permite integrar módulos, contenedores y plug-ins. La figura 4 muestra como se integran los diferentes módulos y el rol de JMX. El grado de modularidad beneficia directamente a los desarrolladores puesto que permite conectar o desconectar componentes de acuerdo a las necesidades. Por ejemplo, si ya no es necesario manejar EJB en la aplicación, simplemente se "desconecta" esta característica y se elimina del servidor. Otro ejemplo es la libertad de poder descargar todo el mapeo de la base de datos objeto relacional dentro del contenedor del servidor.

JBoss emplea una arquitectura de precarga de clases que facilita la comunicación entre las unidades de desarrollo y entre los servicios de la aplicación. JBoss también cuenta

[6] The JBoss JMX integration bus and the standard JBoss components.
http://www.ubookcase.com/book/Sams/JBoss.4.0.The.Official.Guide/0672326485/ch02lev1sec1.html

con una arquitectura de despliegue extensible, esto significa que es permitido adicionarle componentes en el **Microkernel JXM.**

JBossNS es una utilidad de nombramiento basado en JNDI,. Este servicio juega un papel importante en la estructura de JBoss ya que permiten al usuario mapear un nombre a un objeto o servicio y así facilitar su búsqueda. JNDI es un API de Java que provee una interface común para una gran variedad de servicios existentes como: DNS, LDAP, Active Directory, RMI registry. Y esta dividido lógicamente en el API del cliente y el SPI *Service Provider Interface*, que permite al usuario crear implementaciones de JNDI para nombrar servicios.

JBossTX es otro servicio importante es encargado del manejo de las transacciones en JBoss. La arquitectura de JBossTX, permite al usuario implementar un gestor de transacciones para cualquier *Java Transaction API* (JTA). Recordemos que una transacción es una unidad de trabajo que puede contener una o más operaciones. Donde las cuatro características de una transacción son: atomicidad, consistencia, aislamiento y duración.

Otro servicio de JBoss es que provee un **contenedor de EJB** implementado a manera de plug-in. De tal manera que un desarrollador puede modificar los aspectos claves del contenedor de EJB y crear una versión personalizada. El contenedor de EJB en un componente encargado de gestionar determinado tipo de clases EJB.

JBoss Messaging o JBossMQ proporciona a JBoss, el JMS (*Java Message Service*) utilizado para el envió de mensaje entre aplicaciones. Los mensajes no se envían directamente a las aplicaciones si no a sus destinos.

JBossCX proporciona a JBoss la arquitectura necesaria para la utilización de JCA (*J2EE Connector Architecture*). JCA es una API cuyo objetivo es estandarizar el acceso a los recursos no-relacionales del mismo modo que JDBC estandariza el acceso a los datos relacionales.

JBossSX es el componente de JBoss que maneja este campo, la seguridad es fundamental ya que se necesita restringir el acceso a las aplicaciones y controlar las operaciones que puede realizar cada usuario.

Existen 2 mecanismos de seguridad

- Seguridad J2EE. Definida en su especificación
- JAAS (Java Authentication & Authorization Service). Permite proteger los recursos restringiendo el acceso a aquellos usuarios que no tienen permisos. Proporciona una capa de abstracción entre la aplicación y sus subyacentes mecanismos de seguridad.

Hibernate, es un popular servicio de persistencia que proporciona una simple, pero poderosa, alternativa a los “beans” estándares. Facilita la creación de clases de persistencia utilizando el lenguaje Java, incluyendo la asociación, herencia, polimorfismo y composición.

Clustering, un cluster es un conjunto de nodos. En un cluster JBoss, un nodo es un servidor JBoss. El clustering permite ejecutar una aplicación en paralelo. La carga se realiza en diferentes servidores, por lo que si alguno falla, la aplicación continua siendo accesible mediante los otros nodos del cluster. Por esto, se puede mejorar el funcionamiento añadiendo más nodos al cluster.

JGroups proporciona el servicio peer-to-peer para que se puedan llevar a cabo comunicaciones entre nodos que formen parte de un mismo cluster

JBossCache es un servicio Diseñado para almacenar en caché los objetos Java a los que se accede más frecuentemente de manera que se aumente el rendimiento de las aplicaciones. Eliminando accesos innecesarios a la base de datos, reduce el tráfico de red e incrementa la escalabilidad de las aplicaciones

JBoss Portal es una Plataforma de código abierto para albergar y servir una interfaz de portales Web, publicando y gestionando el contenido así como adaptando el aspecto de la presentación.

JBoss Forum proporciona un servicio de foro. Es un foro de discusión en Java, soporta base de datos MySQL, PostgreSQL y HSQLDB, una interfaz altamente configurable, soporta un número ilimitado de grupos de usuarios con permisos distintos, notificaciones por email de actividad en los post, etc.

JBoss AOP es otro de los módulos que JBoss, por medio del cual soporta es el desarrollo orientado a aspectos. JBoss AOP es un marco de trabajo construido en Java orientado a aspectos, que puede ser usado en cualquier ambiente o integrado con el servidor JBOSS AS. El uso de aspectos permite manejar una mayor modularidad que la programación orientada a objetos. También ofrece una separación más limpia de la lógica y del código del sistema. En la siguiente sección explicaremos mas detalladamente cada una de las funcionalidades del framework aspectual de JBoss, el JBoss AOP.

3. 2 INTRODUCCION AL FRAMEWORK JBOSS AOP

3.2.1 Presentación de La Herramienta

El JBoss soporta programación orientada a aspectos a partir de la versión JBoss AS 4.0. JBoss AOP es un proyecto del grupo JBoss Professional Open Source[7]. El enfoque de orientación a aspectos introdujo nuevos conceptos para los desarrolladores y para las aplicaciones empresariales. Ya que AOP extiende la tradicional programación orientada a objetos para mejorar la reutilización de código a través de diversas jerarquías de objetos.

El concepto básico en AOP es el Aspecto, que es un comportamiento común que se encuentra claramente identificable en diferentes métodos, de clases, de jerarquías de objetos, y a veces incluso en modelos enteros de objetos. Así como los componentes básicos de la programación a objetos OOP son: la herencia, el encapsulamiento y el polimorfismo, los elementos básicos de la programación orientada a aspectos son: los consejos/interceptores (*advices/interceptors*), introducciones (*introductions*), metadatos (*metadata*) y los puntos de corte (*pointcuts*).

Los consejos (advices) son una lógica que es dispara por cierto evento. Este comportamiento particular puede ser insertado entre el objeto que invoca y el objeto que es invocado. Por decirlo de otra forma entre un método invoca() y el método actual invocado(). Este elemento es el que permite definir los comportamientos cruzados dentro de la funcionalidad. En JBoss AOP se implementan los consejos usando interceptores (*interceptors*). De tal forma que uno puede definir consejos que intercepten métodos invocados, constructores o propiedades de una clase. Este comportamiento lo veremos detalladamente mas adelante.

Las introducciones (*introductions*) permiten al usuario adicionar métodos o propiedades a una clase existente. También por medio de las introducciones podemos intervenir y

[7] http://labs.jboss.com/

cambiar una interfaz que actualmente esta siendo usada por una clase e introducir una nueva clase mezclada (*Mix-in class*) que implementa la nueva interfaz que modificamos. Otra función de las introducciones es que permiten implementar herencia múltiple en las clases Java.

Los metadatos son información adicional que puede ser adjuntada a una clase, puede ser en tiempo de compilación o en tiempo de ejecución. Su mayor potencial esta cuando son usados para adicionar dinámicamente información para determinada instancia de un objeto. Una analogía de los metadatos muy usado actualmente son las especificaciones de EJB. Los EJB poseen un archivo XML que son los descriptores de despliegue, donde se puede definir método por método, los atributos de las transacciones. De tal forma que el servidor sabe cuando y como comenzar, o cuando suspender, por que anteriormente se han descrito en el archivo XML de configuración.

Los puntos de corte (*pointcut*) son los encargados de integrar los interceptores, las introducciones y los metadatos. Los puntos de corte le dicen al framework JBoss AOP qué interceptores asociar a qué clase, qué metadatos aplicar a qué clase o qué clases serán afectadas por una introducción. Los puntos de corte definen que características de la programación AOP son aplicadas en las clases de una aplicación.

3.2.2 Arquitectura de la herramienta JBoss AOP

JBoss AOP es un framework orientado a aspectos construido en Java. Este framework puede ser usado en cualquier ambiente de desarrollo o integrado al Servidor de aplicaciones JBoss (JBoss AS). En este proyecto vamos a enfocarnos al uso de JBoss AOP integrado con el Servidor de Aplicaciones de JBoss.

Los puntos de corte en JBoss AOP están habilitados para interceptar un campo de lectura y escritura, un método, un constructor, el punto en el que un método llama a otro método, o el punto cuando un constructor llama a un método o a otro constructor.

El modelo de puntos de unión (*joinpoint*) en JBoss AOP es invasivo. Esto significa que JBoss AOP puede interceptar los puntos de unión en la ejecución de la aplicación independientemente si estos son públicos, privados o protegidos.

En JBoss AOP los puntos de corte son descritos separadamente de los aspectos y de los objetos a los que se aplican estos aspectos. Para conservar la independencia, estos puntos de corte son configurados en un archivo XML externo. Los puntos de corte son expresiones regulares que contiene información acerca de las clases que son afectadas por los aspectos, los métodos de estas clases y los parámetros de dichos métodos, al igual que los campos de las clases. Por cada expresión JBoss provee un patrón diferente de construcción, que puede ser:

- Patrón de tipos (*Type pattern)*
- Patrón de método (*Method pattern)*
- Patrón de constructor (*Constructor pattern)*
- Patrón de campo (*Field pattern)*

JBoss AOP ofrece dos maneras de enlazar los puntos de unión y los consejos. Una es usando un archivo XML y la otra forma es usando las anotaciones (*annotations*) de Java 5.0. El XML provee un elemento <*bind*> que describe el enlace entre el punto de corte y el aspecto al que será aplicado el punto de corte. JBoss AOP resuelve los enlaces entre los puntos de corte y los consejos en tiempo de ejecución.

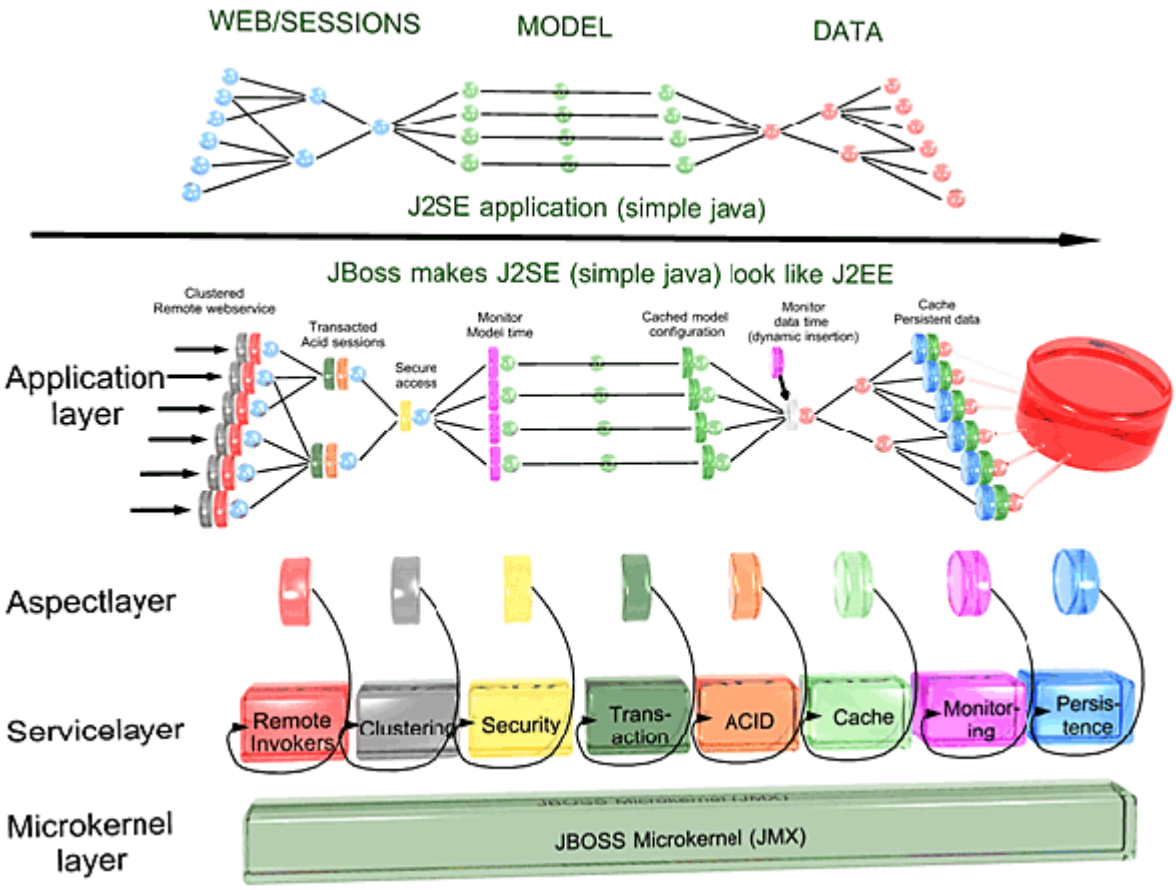

Figura 5: Integración de JBoss AOP a la arquitectura de JBoss AS [8]

De la figura 5 podemos observar como los aspectos se encapsulan la funcionalidad de los servicios y proveen una capa superior que se puede integrar más fácilmente con la capa de aplicación.

La capa de microkernel es el núcleo del servidor de aplicaciones ya que provee toda la integración de los servicios a través de JMX. Esta es la capa encargada de realizar la carga de las clases y de la funcionalidad de cada una de ellas en tiempo de ejecución, además de la gestión de todo el ciclo de vida de cada uno de los servicios.

[8] Arquitectura de JBoss AOP. http://www.jboss.org/products/jbossas/architecture

La capa de servicios reside arriba de la capa del microkernel y tiene una Arquitectura Orientada a Servicios, (SOA- *Services-Oriented Architecture*). Esto consiste en una serie de servicios que se empaquetan cuidadosamente en tiempo de despliegue. Los servicios de JBoss abarcan toda la capa de servicios J2EE. Existen servicios de manejo de transacciones, servicios de mensajería, servicios de seguridad, gestión de las conexiones a las BD. Fácilmente el usuario puede agregar o retirar servicios innecesarios para el despliegue de las aplicaciones, para mejorar el rendimiento. Además de esto, el usuario esta capacitado para construir sus propios servicios y desplegarlos como SARs (Service Archive) contenidos en el servidor de aplicaciones de JBoss. Donde cada servicio puede ser desplegado individualmente.

La capa de aspectos esta basada en el modelo de Programación Orientada a Aspectos (AOP, Aspect-Oriented Programming). Esta modularidad esta soportada por el concepto y el manejo de interceptores, ya que se pueden quitar o agregar los interceptores que son los encargados de enlazar el comportamiento de proporcionado por los servicios.

La capa de aplicación es donde residen las aplicaciones del usuario. Estas aplicaciones pueden utilizar cada uno de los servicios que provee el servidor de aplicaciones JBoss.

3.2.3 Mecanismos para el tratamiento de aspectos

Configuración Clásica usando archivos XML: Esta es la más común y al mismo tiempo la más sencilla de las formas en las que se puede configurar el uso de aspectos en una aplicación de software. Consiste en definir en un archivo XML, el tejido entre las clases del código fuente con las clases de los aspectos, para el caso de JBoss AOP, este archivo se llama jboss-aop.xml. Dependiendo del framework aspectual utilizado pueden cambiar o no, las palabras de los tags del archivo de configuración.

Anotaciones: Anotaciones o en ingles Annotations es un nuevo concepto que ha sido introducido a partir de la versión de Java, JDK 5.0. Las anotaciones puedes ser otra alternativa para configurar las clases aspectuales en JBoss AOP además de los

archivos XML. Por ejemplo para marcar una clase como aspectual se debe utilizar la anotación `@Aspect`.

MBeans o JMX: Viene de las palabras en ingles Managed Beans, que son los componentes que pueden ser administrador a través el Servidor JMX, también conocido como servidor de MBeans. Un MBean consiste en que expone una interface que puede ser utilizada por el Servidor de MBean para llamar los métodos en el MBean. En otras palabras los aspectos pueden ser manejados como un servicio de la aplicación dentro del nucleo de JBoss AS, que es el Servidor JMX.

4. PRESENTACIÓN DEL CASO DE ESTUDIO

4.1 DESCRIPCIÓN GENERAL DEL SISTEMA POS

Los negocios cuya actividad económica es la venta de gran número de productos y variedades de manera directa a sus clientes, requieren de un sistema que les permitan mantener un control permanente sobre las ventas, el inventario y la interacción con sus proveedores y clientes. Debido a la gran variedad de negocios que basan su actividad económica en la compra, venta y distribución de productos, los sistemas POS "*Point of Sale*" tienen diferentes alcances según el tipo de negocio particular al cual se aplica: Supermercados y autoservicios, tiendas de negocio de venta minorista, gastronomía [9].

El tipo de pos que va a ser especificado cae en la categoría de tiendas de negocio de ventas minorista. Este tipo de POS esta orientado al sector de pequeños negocios que requieren de un manejo más amigable y sencillo de los movimientos que genera la actividad comercial. La aplicación debe remitirse estrictamente a los requisitos para no involucrar funcionalidad no requerida, esto hace que los proveedores de aplicaciones liberen diferentes versiones del producto, las cuales incluyen funcionalidades acorde con las particularidades del negocio.
El caso de estudio que se utilizó es una versión reducida de un sistema POS, que maneja compras y ventas.

4.2 MODELO CONCEPTUAL DEL SISTEMA POS

[9] Alexander Barón, Análisis Comparativo AOSD/UC y Theme/UML. Reporte técnico, grupo de Ingeniería de Software Universidad EAFIT, 2007

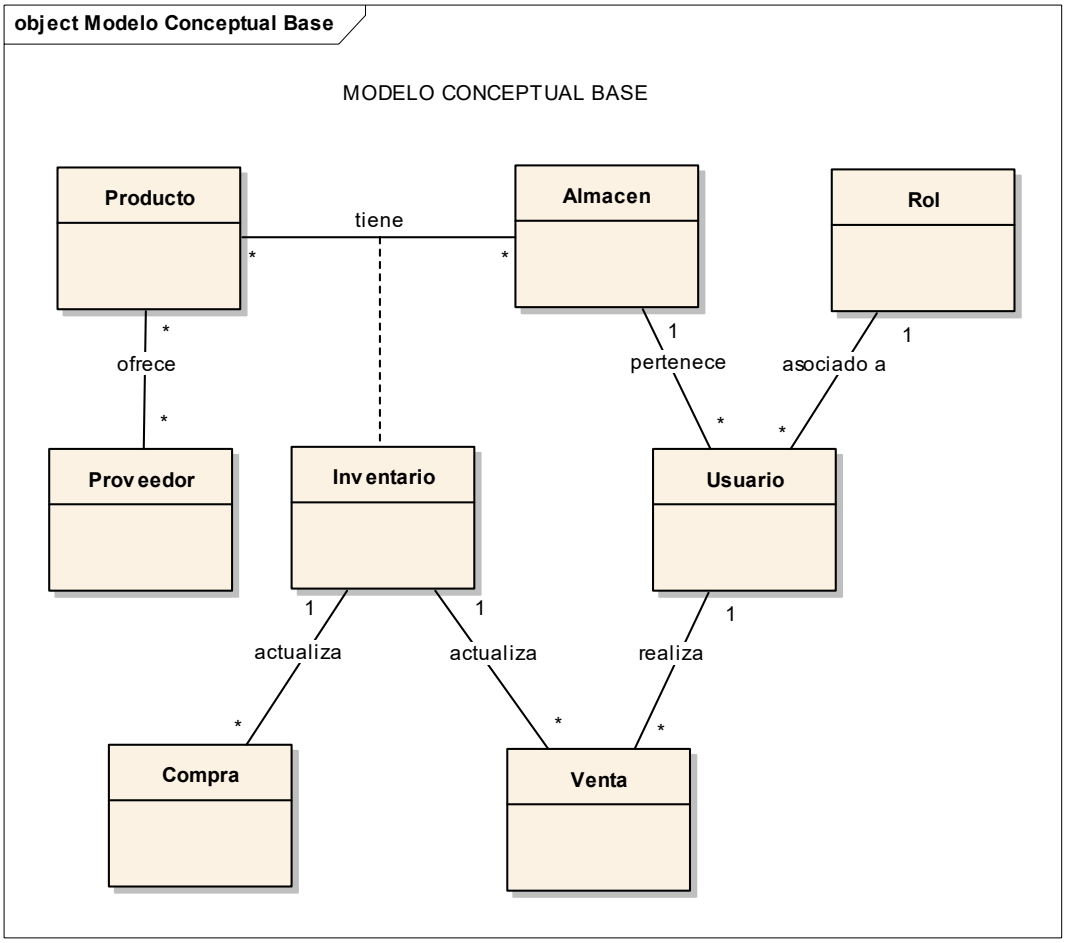

Figura 6: Modelo Conceptual Base

4.3 PAQUETES DE FUNCIONALIDAD BASE

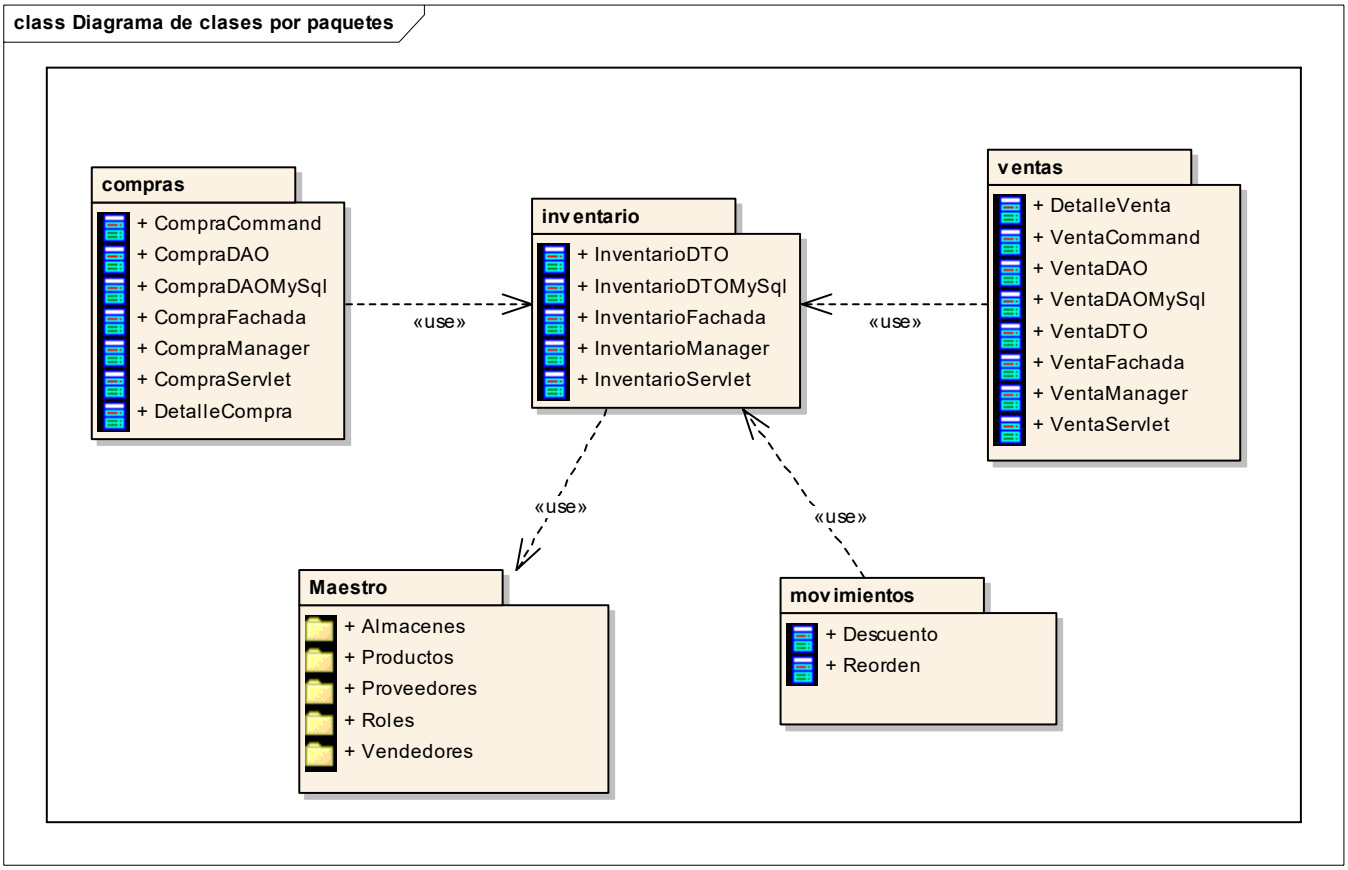

Figura 7: Diagrama de clases por paquetes

4.4 ARQUITECTURA DE REFERENCIA

Se trata de un sistema Web construido en Java que utiliza una arquitectura multinivel en la que se distinguen las siguientes capas:

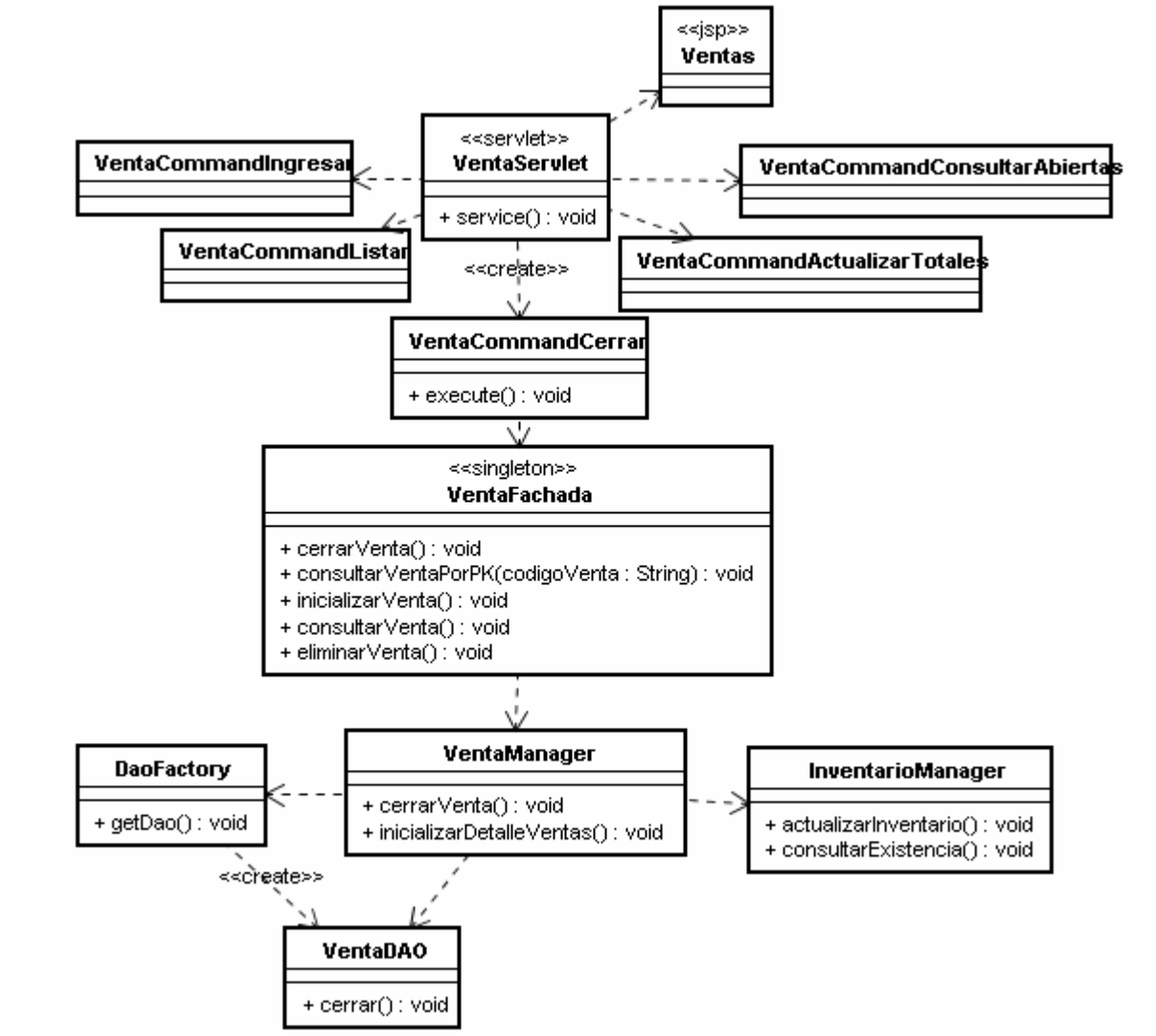

Figura 8: Clases involucradas en la operación cerrarVenta

Capa de presentación: Representa la interacción con el usuario. Implementado a través de archivos Java Server Pages, JSP.

Capa de control: Conjunto de clases que reciben la petición del lado del cliente en clases tipo Servlet y se encargan de invocar el comando específico asociado a la operación.

Capa de lógica de la aplicación: Representada por medio de clases tipo fachada que se encargan de concentrar todos los servicios de aplicación y hacer el manejo de excepciones que se reciben de las capas internas.

Capa de lógica de negocio: Representa la lógica de objetos de negocio que ofrecen servicios generales a más de una aplicación. Un objeto de negocio complejo estará representado por un objeto manager que es el responsable de coordinar la interacción con los demás objetos de negocio y los respectivos objetos que representan el estado del objeto de negocio.

Capa de transferencia de datos: Esta capa mantiene el estado de los objetos de negocio. Los objetos de transferencia de datos (xxxDTO – Data Transfer Object) son los objetos que se comparten a través de las diferentes capas de la arquitectura con el propósito de realizar una operación específica.

Capa de acceso a datos: Es la capa encargada de realizar los servicios de persistencia a la base de datos, utilizando el patrón DAO.

5. MODELADO DE ASPECTOS

5.1 ASPECTOS A IMPLEMENTAR

En este trabajo se planea implementar algunos aspectos representativos que ilustren la manera como esta aproximación puede articularse dentro de una aplicación web. La figura numero 9 presenta la estructura de módulos donde se puede observar los módulos base y sus relaciones de dependencia y los módulos aspectuales junto con las relaciones de intercepción o corte (<<crosscut>>) sobre la funcionalidad base. A continuación se presenta el diseño de cada uno de los aspectos siguendo el enfoque de JBossAOP. La primera versión de la aplicación implementó los tres primeros aspectos.

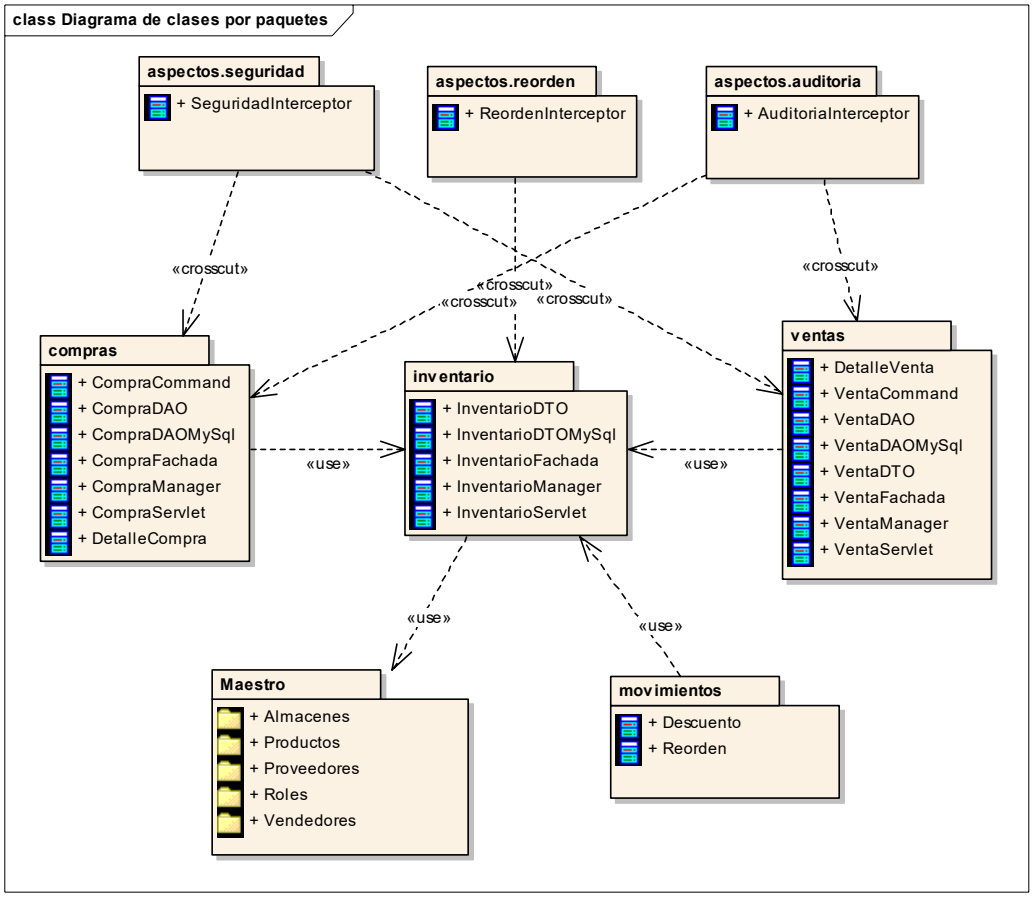

Figura 9: Diagrama de clases por paquetes

5.2 ASPECTO DE AUTENTICACIÓN

La autenticación es una de las maneras básicas como se puede implementar la seguridad. Una implementación no aspectual de la autenticación obliga a que cada opción de menú que se invoca y que requiere control de acceso, realice de manera explícita el llamado a un servicio de autenticación.

En este caso de estudio, el aspecto de autenticación detecta la invocación del servicio del servlet en el que se activa el llamado desde la capa de presentación (service(HttpServletRequest request, HttpServletResponse response). Ver figura 10. En el caso de que se trate de un servicio que requiere control de acceso, mostrará la ventana de login y password (clase SeguridadServlet). Si el usuario ya ha sido autenticado en un proceso previo, el servicio se encargará de verificar que tenga el privilegio requerido para realizar la operación. Asimismo el aspecto actualiza un objeto (ContextoAplicacion) que guarda la información de la sesión activa, la cual será utilizada por otros aspectos.

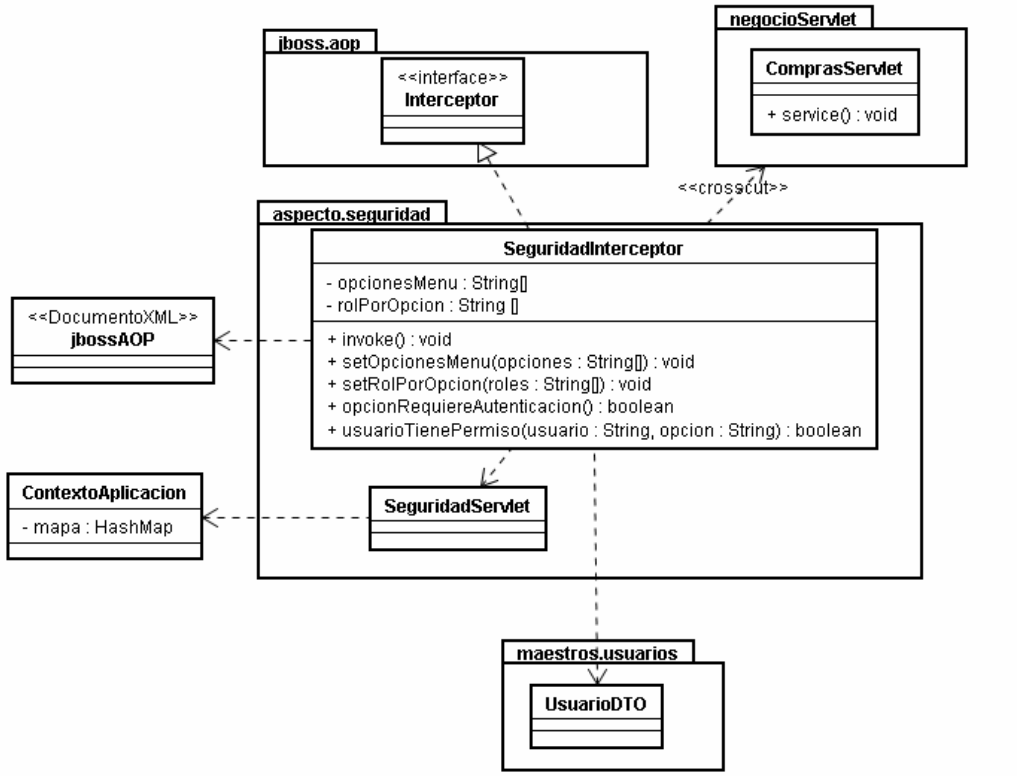

Figura 10: Diseño del aspecto de autenticacion

La clase que sirve de entrada a la implementación de aspecto (SeguridadInterceptor), debe implementar la interfase jboss.aop.Interface. En el archivo de configuración (jbossAOP.xml) se definen las operaciones que requieren control de acceso (opcionesMenu) y el rol respectivo que tendrá privilegio para realizar dicha operación (rolPorOpcion), como puede observarse en la figura numero 11. Estos datos son cargados a la estructura respectiva utilizando las facilidades de reflexión que posee Java. El pointcut del aspecto se define de manera genérica como “intervención del método llamado service correspondiente a cualquier clase cuyo nombre termina en Servlet y que se encuentra ubicado en cualquier paquete dentro del path proyecto.pos”.

```
<aspect class="proyecto.pos.aspectos.seguridad.SeguridadInterceptor">
    <attribute name="opcionesMenu">CompraServlet,VentaServlet</attribute>
    <attribute name="rolPorOpcion">ADMIN,ADMIN</attribute>
</aspect>

<bind pointcut="execution(public void proyecto.pos.*.*Servlet-&gt;service(..))">
    <interceptor class="proyecto.pos.aspectos.seguridad.SeguridadInterceptor" />
</bind>
```

Figura 11: Configuración del aspecto de autenticación

El aspecto se encarga de acceder a la información de usuarios y roles para determinar si el usuario que ha ingresado tiene el rol requerido por la operación que se activó.

5.3 AUDITORIA DE OPERACIONES CRÍTICAS

Para algunas operaciones de actualización sobre la base de datos, es importante generar un rastro de auditoria. Dicho registro debe asociar información acerca del usuario activo y el detalle correspondiente al tipo de objeto específico que se esta auditando. En este aspecto fue posible ilustrar la asignación de comportamientos adicionales a una clase definida en la funcionalidad base y que se conoce como introduction. Se requiere que los objetos que se están auditando (VentaDTO y CompraDTO) tengan un comportamiento adicional que es requerido para propósitos de la auditoria y que se definen en la interface IAuditoriaTrace: getNombreTabla y getDetalle; estos servicios son implementados por las respectivas clases Mixing Ver figura numero 12.

La auditoria hace uso de la información almacenada en el contexto de la aplicación para obtener información de usuario activo en la sesión. Esta estrategia de conservar un espacio global en el que se guarde información de la capa de control, permite que el aspecto que opera sobre una capa interna pueda tomar información de las capas superiores, sin que dichos parámetros tenga que pasarse a través de la cadena de llamados.

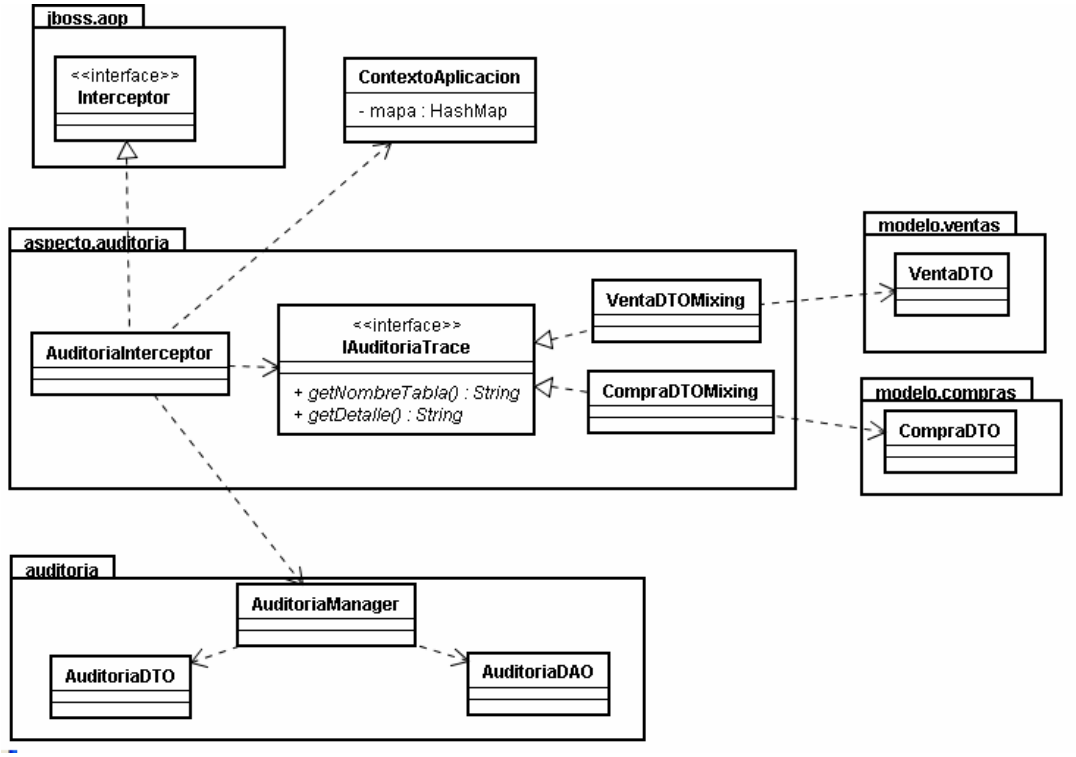

Figura 12: Diseño de aspecto de auditoria

La clase AuditoriaInterceptor es la encargada de recibir la notificación del joinpoint; al igual en el aspecto anterior, la auditoria crea clases que permiten llevar a un medio persistente el rastro de auditoria. Como se puede observar en la configuración de aspecto Ver figura 14, se interviene el servicio ingresar de las clases CompraDaoMySql y VentaDaoMySql los cuales recibirán como parámetro el objeto DTO respectivo. El aspecto hará uso del comportamiento agregado de las clases respectivas a través de la interface IAuditoriaTrace. La figura yy muestra la manera como se definen las introduction en el archivo de configuración

```
<aspect class="proyecto.pos.aspectos.auditoria.AuditoriaInterceptor" />
<bind pointcut="execution(public * proyecto.pos.compras.CompraDAOMySql-&gt;ingresar(..))">
    <interceptor class="proyecto.pos.aspectos.auditoria.AuditoriaInterceptor" />
</bind>

<bind pointcut="execution(public * proyecto.pos.ventas.VentaDAOMySql-&gt;ingresar(..))">
    <interceptor class="proyecto.pos.aspectos.auditoria.AuditoriaInterceptor" />
</bind>
```

Figura 13: Declaración de aspecto de auditoria

```
<introduction class="proyecto.pos.compras.CompraDTO">
    <mixin>
        <interfaces>
            proyecto.pos.aspectos.auditoria.IAuditoriaTrace
        </interfaces>
        <class>proyecto.pos.aspectos.auditoria.CompraDTOMixin</class>
        <construction>new proyecto.pos.aspectos.auditoria.CompraDTOMixin(this)</construction>
    </mixin>
</introduction>

<introduction class="proyecto.pos.ventas.VentaDTO">
    <mixin>
        <interfaces>
            proyecto.pos.aspectos.auditoria.IAuditoriaTrace
        </interfaces>
        <class>proyecto.pos.aspectos.auditoria.VentaDTOMixin</class>
        <construction>new proyecto.pos.aspectos.auditoria.VentaDTOMixin(this)</construction>
    </mixin>
</introduction>
```

Figura 14: Configuración de introductions para las clases CompraDTO y VentaDTO

5.4 COLOCACIÓN AUTOMÁTICA DE PEDIDO

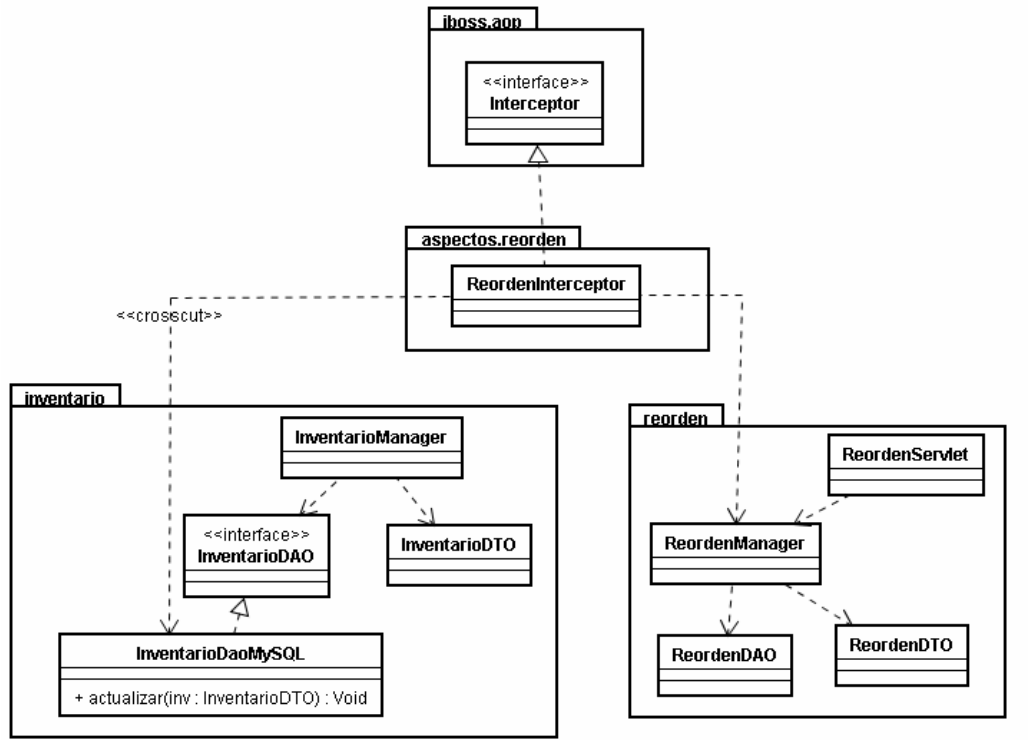

Figura 15: Diseño del aspecto de reorden

Hasta el momento, los dos aspectos que se han explicado están orientados a implementar funcionalidades asociadas a atributos de calidad, que en este caso es la seguridad. Interesa también analizar la aplicación del enfoque de aspectos para el manejo de características funcionales del sistema. Uno de los ejemplos elementales en el sistema POS es el manejo automático de pedido. Cada vez que se hace una salida de producto del inventario se chequea si la nueva existencia se encuentra por debajo de la cantidad mínima en inventario, en cuyo caso genera un registro de pedido. Igualmente cada vez que se realiza el proceso de conteo de productos existentes en el almacén y se registra la existencia real, si la existencia se encuentra por debajo de la cantidad mínima se genera un registro de pedido.

Como puede observarse en la figura 16, el aspecto ReordenInterceptor interviene la operación actualizar de InventarioDaoMySQL y compara la nueva existencia con el punto de reorden. Si la nueva existencia se encuentra por debajo del punto de pedido ingresa un registro de pedido a través de la clase ReordenManager.

5.5 APLICACIÓN DE DIVERSOS DESCUENTOS

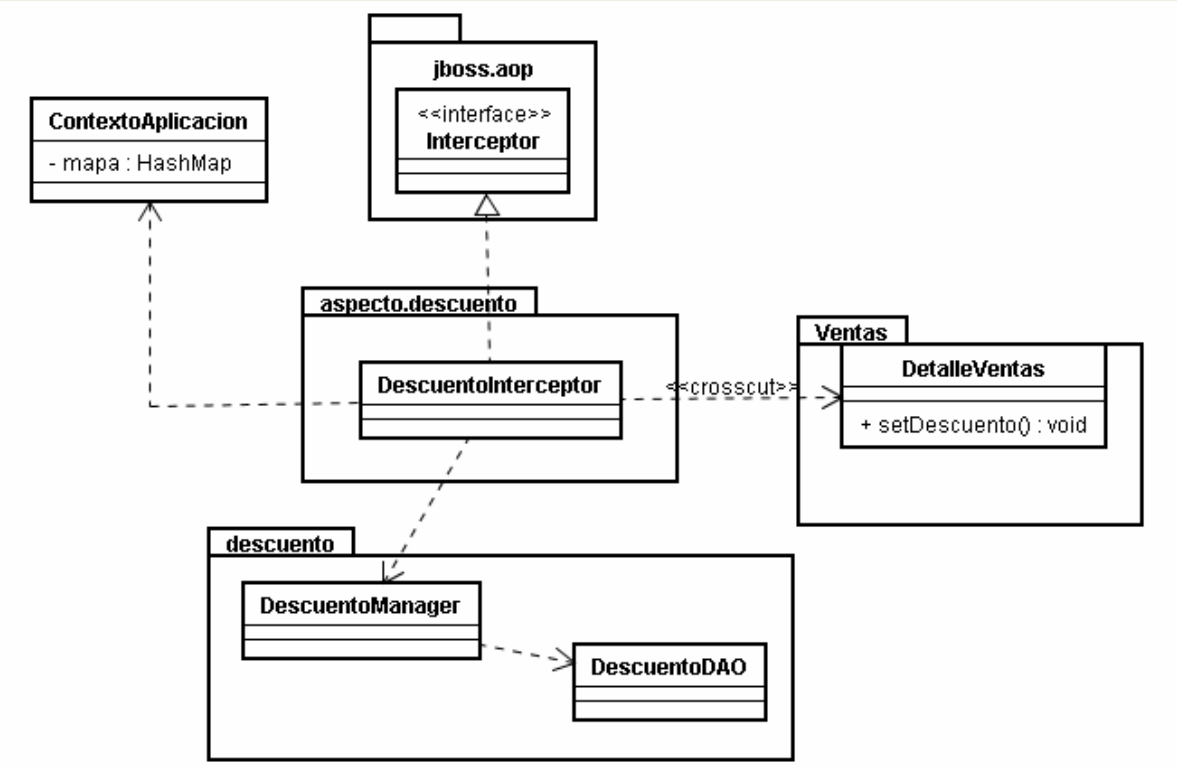

Figura 16: Diseño del aspecto de descuento

En un sistema POS es muy frecuente que se realicen descuentos sobre algunos productos. Además se pueden tener diversas estrategias para manejo de descuentos, como por ejemplo, descuentos por productos concretos, descuentos por fechas especiales o promociones especiales. Esto hace suponer que esta funcionalidad puede diseñarse de manera aspectual. Una características particular de este aspecto es que su comportamiento se hace visible en la presentación cada vez que se agrega

un item de la venta; es por esta razón que la clase DescuentoInterceptor debe acceder a las variables del ContextoAplicacion con el propósito de intervenir la información que va a ser manejada por el jsp. Los descuentos generados se almacenan en una bitácora de descuentos por medio de la clase DescuentoManager.

5.6 MANEJO DE MÉTRICAS

Otro uso interesante que se le puede dar al aspecto es como funcionalidad que sirve para perfilar el comportamiento de un aplicativo para determinar si se cumple con una condición mínima de tiempo de respuesta en ciertas operaciones que por su naturaleza se consideran críticas. Este aspecto es útil en la fase de pruebas y se puede eliminar del producto en operación tan solo cambiando el archivo de configuraciones, sin que se tenga que cambiar el código de la aplicación.

Para el sistema POS puede ser conveniente realizar monitoreo de ciertas operaciones críticas para detectar si los tiempos de respuesta superan un valor máximo permitido. Uno de los puntos específicos en los que el monitoreo puede resultar útil es en la operación que se encarga de mostrar la ventan emergente de la lista de productos desde donde se selecciona el producto que va a ser vendido. Puesto que esta ventaja emergente recorre todos los productos existentes, es posible que dicha operación requiera un proceso de optimización adicional.

En el archivo de configuración del aspecto se puede definir el tiempo máximo de respuesta permitido. El aspecto toma registro del tiempo de inicio antes de ejecutar la operación y toma el registro de tiempo después de la operación. Si el tiempo de ejecución del evento supera el tiempo máximo permitido, se genera un registro en el archivo de errores indicando que no se supera el tiempo acordado.

5.7 ESPECIFICACIÓN RESUMIDA DE ASPECTOS ESTUDIADOS

Se han detallado algunos de los aspectos que fueron diseñados para el sistema POS. En la Tabla numero 1 presenta un resumen de las principales características encontradas en los aspectos estudiados

Aspecto	Capa que interviene	Pointcut	Alcance
Autenticación	Control	*Donde*: *Servlet.services(..) *Cuando*: Antes	Capa de presentación: Ventaja de login y password Capa de acceso a datos: Para consultar rol del usuario activo. La definición de rol por opción es dada por medio del archivo que configura el aspecto Publicación de la sesión en *ContextoAplicacion* (Singleton)
Auditoria	Acceso a datos	*Donde:* *DaoMySQL.ingresar(..) *Cuando:* Despues	Extender el comportamiento de **DTO* para ofrecer servicios requeridos por la auditoria. Las respectivas clase **DTOMixin* implementan la interfase *IAuditoriaTrace* Capa de acceso a datos: Para ingresar un registro en la tabla de bitácora de auditoria
Reorden	Acceso a datos	*Donde:* InventarioDaoMySQL.actualizar(..) *Cuando:* Despues	Capa de acceso a datos: Para ingresar un registro en la tabla de bitácora de auditoria Consultar las reordenes colocadas por el sistema automáticamente
Descuentos	Lógica Negocio	*Donde:*DetalleVenta.setDescuento() *Cuando:* Despues	Capa de presentación para mostrar el descuento que se aplicó Capa de acceso a datos para generar un registro de descuentos

Métricas	Lógica de control	*Donde:*método que dispara el servicio de la ventaja emergente *Cuando:* Alrededor	Capa de acceso a datos para registrar en un archivo las operaciones que no cumplen el mínimo del tiempo requerido

Tabla 1. Resumen de aspectos estudiados

6. EVALUACIÓN DE JBOSS AOP

En esta sección se realiza la evaluación de JBoss AOP según los criterios establecidos por el Grupo de Ingeniería de Software de Universidad Eafit y que ha servido de referentes para evaluar otros frameworks (López, 2007) (Hincapié, 2007). El significado de algunos criterios tuvo algunos cambios con respecto al significado que se le dio en los trabajos anteriores, con el propósito de buscar una aplicación más práctica del criterio.

6.1 CRITERIOS GENERALES

En la Tabla numero 2 muestra el resultado de la evaluación cuantitativa, de acuerdo a los criterios que se describen en el documento principal. Cada calificación viene acompañada con su respectiva justificación y obedece a la escala de valores ya establecida.

6.1.1 Escalas

Para esta segunda fase se utilizó una escala cuyos valores fueron asignados por juicio de un experto de arquitectura. En general estos valores corresponden a 5 si soporta completamente la característica y 1 si no lo soporta; algunos de los criterios justificaron la definición de un valor intermedio.

#	Criterio	Justificación	Calificación
1	Desempeño	La inclusión de aspecto en un comportamiento no presenta un tiempo mayor significativo. De otra parte, se exploro el apoyo que los aspectos pueden proveer al desempeño del aplicativo: a través de un aspecto se puede monitorear el tiempo empleado por algunas operaciones que se consideran críticas, sin que haya necesidad de hacer cambios en la	5

		funcionalidad base.	
2	Seguridad	Puesto que JBoss AOP es un frameworks basado en Java, es posible integrarlo con plataformas como JAAS (Java Authentication and Authorization Service). Para el caso de estudio se implementó la funcionalidad de autenticación de manera aspectual; esto podría llevarnos a pensar que, si bien se podría seguir utilizando servicios como los provistos por JAAS es posible que con aspectos, el llamado a dichos servicios se pueda hacer de forma aspectual.	5
3	Integración con plataforma Web	Para la introducción de los aspectos en una aplicación web, es necesario desplegar la aplicación en el Servidor de JBoss. Esta fue una de las tareas más complejas de realizar puesto que fue necesario construir la aplicación a través de ant, para que tuviera la estructura de carpetas necesaria para que funcionaran los aspectos en una aplicación Web, corriendo en el JBoss AS.	3
4	Integración con plataforma Stand Alone	JBoss AOP provee una solución stand alone que funciona como un plugin de eclipse lo cual facilita la definición y aplicación de los aspectos Este entorno permite incluso definir y consultar los aspectos de manera interactiva.	5
5	Flexibilidad	Al igual que en otros frameworks la aplicación o no del comportamiento aspectual se realiza con solo modificar el archivo de configuración donde se define el aspecto. JBoss AOP permite asociar propiedades de las clases que implementan el aspecto por medio del archivo de configuración o utilizando las facilidades de meta datos que provee. Otra forma como se puede ver la flexibilidad en JBoss AOP por medio de clases tipo Mixing las cuales extienden el comportamiento de una clase (por ejemplo un objeto de negocio) con la introducción de una interfaz de forma dinámica.	5
6	Capacidad para realizar DEBUG	No se evaluó esta capacidad del framework JBoss AOP	-
7	Herencia	En JBoss AOP el aspecto se define por medio de una clase de Java que implementa una interfase determinada (método *invoke* de la interfase *jboss.aop.Interceptor*). Esto permite deducir que en la lógica del aspecto se pueden utilizar todas las ventajas de herencia de Java entre las clases que representan el	5

		aspecto.	
8	Separación alcanzada	No esta clara la manera como podría analizarse esta característica en JBoss AOP.	-
9	Reusabilidad	A través de la definición de “willcards” el aspecto puede ser reutilizado en diferentes puntos del sistema. JBoss AOP permite también definir orden de precedencia de la aplicación de diferentes aspectos en un mismo punto del sistema.	5

Tabla 2. Resultados Evaluación Cuantitativa

6.1.2 Curva de aprendizaje

El sitio oficial de JBoss AOP contiene un buen número de material que facilita el estudio de la herramienta. El plugin desarrollado en Eclipse nos facilito las pruebas de concepto iniciales. Los ejemplos mostrados en la guía de usuario fueron suficientemente ilustrativos para implementar los aspectos en el caso de estudio. Fue también de bastante utilidad los resultados de una tesis de maestría (Dalager, et.al 2004) que nos ayudo a entender algunos de los conceptos del framework. Consideramos que con el presente trabajo aun no se ha concluido con la curva de aprendizaje del framework. Al revisar la tesis de maestría se pudo observar, nuevas maneras de implementar los aspectos como servicios asociados al servidor, que por razones de tiempo no fue posible entrar a estudiar. Asimismo se puede observar que las librerías del servidor traen ya la implementación de aspectos cuyo uso requiere un mayor tiempo de aprendizaje.

6.1.3 Características generales

- **Instanciación:** En JBoss AOP el aspecto puede ser definido con diferente alcance: A nivel de máquina virtual (PER_VM) a nivel de clase (PER_CLASS) a nivel de instancia (PER_INSTANCE) o a nivel de Join Point (PER_JOINPOINT).

- **Complejidad:** Consideramos que la introducción de los aspectos en JBoss AOP no implica un aumento considerable de la complejidad. Desde el punto de vista de la arquitectura se tuvo cuidado de declarar en paquetes independientes la funcionalidad que representa el aspecto, lo cual facilita su ubicación dentro del conjunto de clases.

- **Acoplamiento:** Si bien el aspecto trae consigo nuevas clases y métodos, la extensión del aspecto no invade el código base, lo que permite que éste se mantenga totalmente desacoplado de la funcionalidad aspectual.

6.2 CRITERIOS DE LA PLATAFORMA

6.2.1 Madurez

JBoss AOP es un proyecto con mucha actividad actualmente lo que hace suponer que se encuentra en proceso de maduración: desde el mes de mayo del 2006 en que publicó su versión JBoss AOP 1.05, liberó 9 versiones en el mismo año hasta llegar a la JBoss AOP 2.0.0 Alpha 1. En el mes de septiembre del presente año liberó su versión JBoss AOP 2.0.o beta1. El proyecto trabaja bajo licencia LGPL.

6.2.2 Precio

El framework JBoss AOP es software libre y se puede descargar gratuitamente, al igual que su documentación, desde Internet. También son gratuitos algunos sub proyectos que se encuentran en el sitio de Internet del framework.

6.2.3 Soporte

El proyecto ofrece los esquemas de soporte propio de las comunidades de software libre: Se presta soporte a través de los foros bajo un esquema de voluntariado, en los

cuales los usuarios de JBoss AOP y los miembros del proyecto responden preguntas sobre diferentes tópicos.

7. CONCLUSIONES Y TRABAJOS FUTUROS

7.1 CONCLUSIONES

El presente trabajo ha realizado un acercamiento a JBoss AOP, un framework que permite la definición y ejecución de comportamiento aspectual. Consideramos que el proyecto cumplió los objetivos definidos en el trabajo de la siguiente manera:

- **Objetivo 1.** Realizar un acercamiento teórico de las características de los frameworks aspectuales existentes. En el capítulo 2 se estudiaron las características generales de un framework aspectual y se entraron a analizar las características principales de dos de los frameworks estudiados en el proyecto: Spring AOP y Jasco.

- **Objetivo 2.** Estudiar las características del framework JBoss AOP, integrado con JBoss AS. En el capítulo 3 se detallaron tanto las características del servidor JBoss como la arquitectura y características del frameworks JBoos AOP. El anexo B describe la manera como se trabaja con el JBoss AOP en el entorno de desarrollo (JBoss AOP IDE) y el anexo C presenta la manera como se debe configurar y desplegar una aplicación web en el servidor de aplicaciones (JBoss AS).

- **Objetivo 3.** Desarrollar un caso práctico que sirva como referencia para la industria y la academia en el uso del framework JBoss AOP integrado con JBoss AS. El capítulo 4 presenta el caso de estudio del sistema POS que fue utilizado. Inicialmente el proyecto pretendía utilizar un software open source ya construido

llamado Tina POS y sobre este software agregar comportamiento aspectual. Esta alternativa fue descartada debido a que Tina POS era una solución stand alone y nuestro objetivo estaba orientado a implementar aspectos en una aplicación web. Se partió entonces de un caso que ya había sido modelado para las pruebas de concepto de otros frameworks el cual se extendió para acercarlo a un problema más real. El capítulo 5 realiza el modelado de los aspectos que fueron diseñados y se hace una explicación de la manera como fueron implementados en JBoss AOP. Se buscó que los aspectos ayudaran en la implantación tanto de características funcionales (como reorden y descuento) como características asociadas a atributos de calidad (seguridad, desempeño).

- **Objetivo 4**. Realizar una evaluación del framework JBoss AOP de acuerdo a los criterios preestablecidos por el Grupo de investigación de ingeniería de sistemas de la Universidad Eafit. El capítulo 6 realiza el análisis del framework teniendo en cuenta los criterios establecidos en el proyecto.

Consideramos por lo tanto que el trabajo cumplió lo objetivos establecidos. Las principales conclusiones que se pueden destacar de este trabajo son las siguientes:

Alcance de los aspectos. JBoss AOP es un framework que permite implementar tanto aspectos asociados a la funcionalidad como aspectos asociados a propiedades sistémicas.

Herramientas de apoyo. El IDE provisto como un plug-in de eclipse ayuda notablemente a entender la manera como JBoss maneja los aspectos. Sin embargo, uno de los problemas encontrados con dicho entorno es que sus diálogos por medio de wizard, los cuales actualizan el archivo de configuración, se encuentran un tanto limitados y por lo tanto no permiten configurar aspectos más complejos. Nos resultó entonces más cómodo entrar a escribir directamente el archivo de configuración con el propósito de describir los aspectos del caso de estudio.

Despliegue de la aplicación. Uno de los principales problemas al que nos enfrentamos fue lograr el despliegue adecuado de la aplicación en el servidor de JBoss. Por que el servidor JBoss AS, necesitaba una jerarquía especifica de carpetas para que reconociera los aspectos en la aplicación Web. La memoria del proceso de despliegue que se siguió se describe detalladamente en el Anexo C. Consideramos que este anexo sirve como guía de instalación para aquellas personas que deseen montar el caso de estudio.

Consideraciones al implementar aspectos en una arquitectura por capas. El diseño de la aplicación siguió una arquitectura por capas, utilizando los lineamientos de la arquitectura de referencia establecidos por la empresa vinculada al proyecto. Los siguientes son algunos de los interrogantes básicos que se deben realizar cuando se introducen los aspectos

- ¿A cuál de las capas de la arquitectura debe intervenir un aspecto? La respuesta depende del alcance que se le desee dar al aspecto. Por ejemplo, en aspecto de auditoria se implementó interviniendo directamente la capa de acceso a datos (clases terminadas en *MySqlDao). Esto significa que la auditoria esta orientada a ser una auditoria de operaciones a la base de datos. Si se desea tener una auditoria orientada hacia servicios de negocio la intervención debería realizar en la capa de lógica de negocio.

- ¿Esta el aspecto limitado a una capa en particular? No. El comportamiento del aspecto bien puede tener incidencia en más de una capa de la aplicación. En el mismo ejemplo de auditoria, el aspecto debe tomar información de las capas superiores para determinar cuál es el usuario activo en la sesión. Es también muy probable que aspectos que intervienen lógica de negocio requieran que su comportamiento se refleje en la capa de presentación. En aplicaciones web esto es un poco complejo puesto que implica intervenir la trama de información enviada como parámetro de la sesión, como se ve en la siguiente línea de código.

  HttpSession session = request.getSession();

- ¿Es posible que más de un aspecto intervenga una misma funcionalidad? Efectivamente, es muy probable que en un mismo punto del sistema se desee aplicar más de un aspecto. En este caso es necesario establecer el orden en que estos aspectos intervienen la funcionalidad base. Para el caso concreto de JBoss AOP esto se declara por medio de una pila.

Madurez de JBoss AOP. Consideramos que el frameworks JBoss AOP es una alternativa madura que puede ser utilizada en proyectos productivos. La confiabilidad de dicha herramienta esta principalmente soportada en el respaldo que tiene este proyecto por parte de su propietario Red Hat. La red mundial de colaboradores que impulsa este proyecto, genera confianza a las empresas que lo utilizan.

7.2 TRABAJOS FUTUROS

Los siguientes son actividades que se podrían realizar para darle continuidad a este trabajo

Implementar los aspectos de descuento y métricas. Por razones de tiempo estos dos aspectos se dejaron a nivel de diseño para que puedan ser implementados posteriormente. La implementación del descuento tiene la variante particular que dicho aspecto debe hacerse visible en la capa de presentación dentro de la línea que muestra el detalle de la venta. Aunque este campo de descuento ha sido previsto desde el diseño inicial de la forma, lo que nos va a permitir el aspecto es un cambio dinámico de si se aplica o no un descuento y, en caso afirmativo, cuál es la estrategia de descuento que se va a aplicar.

Implementar aspectos como servicios provistos por el servidor (JBoss AS). Otra forma en que los aspectos pueden ser configurados es como servicios declarados en el servidor. Consideramos que esta debe ser una alternativa importante que debe ser

evaluada con el propósito de encontrar sus ventajas y desventajas con respecto a la implementación utilizada en el presente trabajo.

Experimentar el uso de anotaciones. Las anotaciones son una alternativa disponible en las últimas versiones de Java con el propósito de colocar marcas en el código fuente para que pueda ser intervenido por un precompilador. Aunque de esta manera, el código original se hace consciente de la intervención de aspecto, consideramos conveniente analizar situaciones en los que es adecuado hacer uso de las anotaciones.

Experimentación de aspectos ya implementados. La versión disponible de JBoss AOP trae consigo un conjunto de clases con aspectos ya implementados. Puede ser muy adecuado entrar a revisar estas librerías de aspectos con el propósito de reutilizar algunas de ellas.

8. BIBLIOGRAFÍA

- H. Ossher, P. Tarr, Multi-dimensional separation of concerns using hyperspaces, IBM Research Report 21452, April, 1999.
- H. Ossher, P. Tarr, Using multidimensional separation of concerns to (re) shape evolving software, Communications of the ACM, October 2001, pp.43-50.
- Booch, G; Rumbaugh, J. y Jacobson, I. El Proceso Unificado de Desarrollo de Software. Addison Wesley, Madrid, 1999.
- David Esteban López. JasCo. Reporte Técnico grupo de Ingeniería de Software, Universidad EAFIT. Agosto-8-2007.
- Jesús Andrés Hincapié. ReporteTecnico SpringAOP. Grupo de Ingeniería de Software, Universidad EAFIT Agosto-8-2007
- AOSD-EUROPE, 2005 Survey of Aspect-oriented middleware. AOSD-EUROPE, 2005.
- Neil Loughran, Nikos Parlavantzas (ULANCS), Monica Pinto, Lidia Fuentes Fernández, Pablo Sánchez (UMA), Matthew Webster, and Adrian Colyer (IBM). Survey of aspect-oriented middleware. Survey D8, AOSD Europe, Jun 14 2005.
- Santiago Henao, Alexander Cañas, David Lozano. Aspect Oriented Middleware Comparison. Reporte técnico grupo de Ingeniería de softwafre, Universidad EAFIT. 2006.

Referencias Virtuales

- The JBoss JMX integration bus and the standard JBoss components. http://docs.jboss.com/jbossas/admindevel326/html/pr03.html

- Documentación acerca de la arquitectura de JBoss AOP. http://www.jboss.org/products/jbossas/architecture
- Sitio de descarga de JBoss AS y JBoss AOP.

  http://labs.jboss.com/projects/download/
- Sitio del proyecto en SourceForge.

  http://sourceforge.net/projects/jboss
- Artículo. Second-generation aspect-oriented programming. Dave Schweisguth.

  http://www.javaworld.com/javaworld/jw-07-2004/jw-0705-aop.html
- Artículo. Bill Burke, Adrian Brock. Aspect-Oriented Programming and JBoss. http://www.onjava.com/pub/a/onjava/2003/05/28/aop_jboss.html
- Documentación JBoss AOP IDE.

  http://docs.jboss.org/aop/1.3/aspect-framework/ reference/en/ html/aopide.html
- Tutorial de Eclipse integrado con Tomcat. http://www.eclipse.org/webtools/jst/components/ws/1.0/tutorials/InstallTomcat/InstallTomcat.html