ARQUITECTURA PARA APLICACIONES WEB ORIENTADA A ASPECTOS BASADA EN LOS CONCEPTOS DE SEPARACIÓN DE INCUMBENCIAS

Monografía para optar al título de Ingeniero en Electrónica y Telecomunicaciones

ALEJANDRA MARÍA NARVÁEZ CAMAYO
OMAR ALBEIRO TREJO NARVÁEZ

Director: **IE. Javier Alexander Hurtado**

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Popayán, Julio de 2.008

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

La programación orientada a aspectos o AOP *(Aspect oriented Programming)* es una tecnología de programación utilizada para encapsular elementos que atraviesan o se encuentran dispersos a lo largo de distintos módulos de una aplicación software. Estos asuntos que afectan a distintos módulos se llaman incumbencias transversales o crosscutting concerns. La solución para encapsular incumbencias transversales en módulos son los llamados *Aspectos,* de tal manera que un aspecto no es otra cosa que una incumbencia transversal modularizada [1].

Un ejemplo típico de incumbencia en una aplicación Web es la autenticación, la cual es requerida por varios métodos de una aplicación que no tienen relación entre sí. En ese caso, la única solución es escribir código repetido que resuelva esa incumbencia para cada subsistema esto produce código disperso (*scattering*) y enredado (*tangling*) [1].

AOP introduce nuevos conceptos que representan comportamientos que se quieren asociar a determinados momentos de la ejecución de un programa, conceptos que no pueden ser soportados por la programación orientada a objetos. Es importante aclarar que AOP complementa la programación orientada a objetos, no la reemplaza [13].

Las aplicaciones Web involucran diferentes elementos que atraviesan transversalmente varios módulos del sistema [31]; una arquitectura orientada a aspectos lo que busca es encapsular dichos elementos y así beneficiar la modularización y reutilización. La arquitectura intenta solucionar el problema de las actuales arquitecturas en las cuales se observa que las incumbencias

transversales atraviesan los elementos arquitectónicos, afectando componentes y conectores [36].

En el presente trabajo se diseña una arquitectura de referencia para la construcción de aplicaciones Web con tecnología orientada a aspectos, que pretende recoger en una única capa los elementos que se encuentran esparcidos a lo largo de un sistema Web y que las arquitecturas tradicionales no logran modularizar. Se identifican los concerns o incumbencias transversales más usuales en una aplicación Web utilizando metodología DSOA (*Desarrollo de Software Orientado a Aspectos*) y se modelan a nivel de paquetes y conectores utilizando la extensión AML (*Aspect Modeling Languaje*), como alternativa a los lenguajes de descripción de arquitecturas *(ADL's)*; estas incumbencias se representan de tal manera que puedan ser reutilizadas y asociadas con los componentes de la lógica del negocio de cualquier aplicación Web que sea construida siguiendo un modelo orientado a objetos en una arquitectura multicapa.

El presente documento está estructurado en cuatro capítulos:

El primer capítulo describe toda la base teórica de los temas que debieron ser aplicados en el desarrollo del proyecto; las generalidades de AOP, metodologías para diseño de aplicaciones y herramientas de implementación.

El segundo capítulo contiene las especificaciones de diseño de la arquitectura de AOP para aplicaciones Web dividida en dos etapas: análisis y diseño, en las cuales se aplican todos los fundamentos vistos en el capítulo anterior y se obtiene un diagrama final arquitectónico que sirve como base para desarrollar aplicaciones Web con AOP.

El tercer capítulo consta de un prototipo de validación de la arquitectura, que en este caso es el portal del Parque Informático Carlos Albán de Popayán, en este capítulo se describen las diferentes decisiones de diseño, implementación y desarrollo, esto involucra selección de la arquitectura que se definió, modelos

descriptivos, diagramas correspondientes, tecnologías empleadas, herramientas de implementación, etc.

El cuarto capítulo contiene las conclusiones respectivas producto de la realización del proyecto, tanto a nivel del desarrollo como a nivel teórico y de utilización de herramientas, de igual manera, contiene las recomendaciones producto de la experiencia adquirida, que puedan ser aplicadas en el desarrollo de trabajos futuros.

Finalmente se presenta la Bibliografía, la cual contiene la referencia a todos los documentos o parte de ellos que fueron consultados como fuente de conocimiento en el desarrollo del proyecto, también se presentan a lo largo del documento una serie de notas y pie de páginas acerca de los términos que deban ser tenidos en cuenta para un mejor entendimiento de la documentación.

I. FUNDAMENTOS DE PROGRAMACIÓN ORIENTADA A ASPECTOS EN APLICACIONES WEB

1.1. Generalidades de la Programación Orientada a Aspectos

La Programación Orientada a Aspectos, más conocida como AOP (*Aspect Oriented Programming)*, es un modelo de programación que busca solucionar un problema específico: capturar funcionalidades de un sistema que los modelos de programación habituales obligan a que estén repartidas a lo largo de distintos módulos del sistema. Estos fragmentos que afectan a distintos módulos son llamados **aspectos** y los problemas que solucionan, incumbencias transversales (*crosscutting concerns*) [1] [2] [5].

AOP encapsula dichos fragmentos en entidades bien definidas, de manera apropiada en cada uno de los casos y eliminado las dependencias inherentes entre cada uno de los módulos. Usando un lenguaje que soporte AOP, se pueden capturar estas dependencias en módulos individuales, obteniendo un sistema independiente de ellos y se pueden utilizar o no sin tocar el código de la operaciones básicas (lógica del negocio) [4].

Las incumbencias transversales [46] son asuntos o intereses que atraviesan transversalmente a otros módulos. Un ejemplo típico de incumbencia transversal para una aplicación Web es la autenticación la cual es requerida en varios puntos de ejecución del sistema que no están relacionados.

El problema aparece cuando una incumbencia afecta a distintas partes del sistema que no aparecen relacionadas en la jerarquía. En ese caso, la única solución suele ser escribir código repetido que resuelva esa incumbencia para cada subsistema esto produce código disperso (*scattering*) y enredado (*tangling*) [4]. Se habla de código disperso cuando un mismo servicio es invocado de manera similar desde

muchas partes o módulos del programa y se habla de código enredado o mezclado cuando en una misma parte o módulo del programa conviven simultáneamente varios servicios o requisitos.

Un ejemplo típico de corte transversal de intereses se presenta en un sistema de despliegue visual posiblemente una GUI[1]. Siempre que una LINEA o un PUNTO sean modificados, se quiere que actualice el despliegue totalmente (ver figura 1.1). Esta es una relación que no puede ser capturada dentro de una jerarquía de herencia en la cual cruzan objetos.

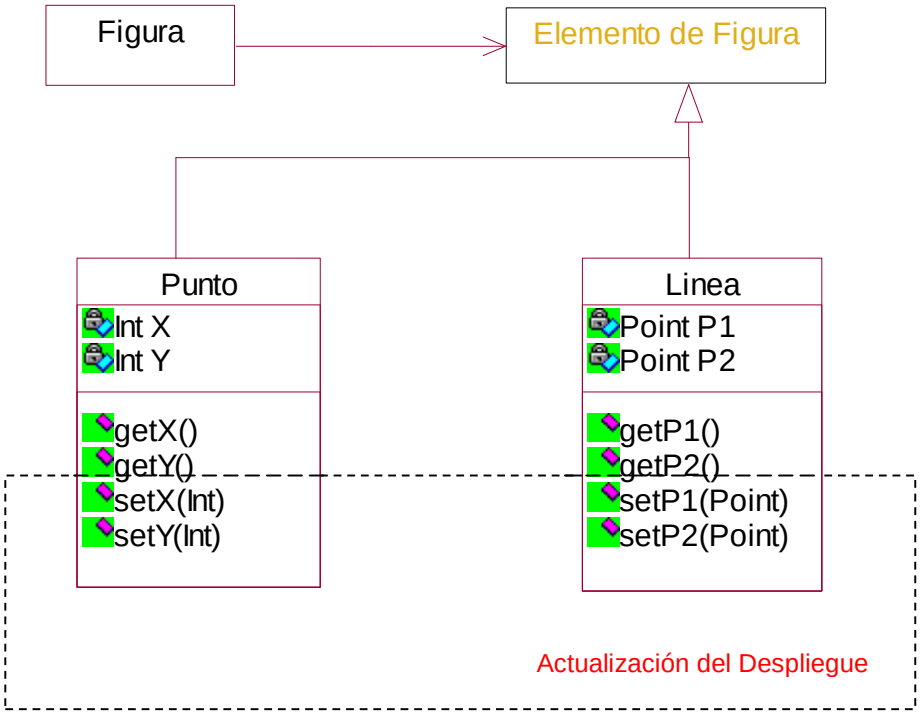

Figura Nº 1.1 Ejemplo de una incumbencia transversal

Los aspectos son la unidad básica de la programación orientada a aspectos. Una definición más formal, y con la que se trabaja actualmente es: "*Un aspecto es una unidad modular que se disemina por la estructura de otras unidades funcionales*[2]".

[1] Graphic User Interface
[2] Definición dada por Gregor Kiczales, 1.999.

De manera pues que un aspecto no es otra cosa que una *incumbencia transversal modularizada*.

1.1.1. Constructores de un Aspecto.

Los constructores que AOP utiliza para especificar las reglas de entrelazado de los aspectos son los siguientes:

- **Join points** (puntos de enlace): Un punto de enlace es un sitio identificable en la ejecución de un programa en los cuales se mezclan los aspectos con el código base. Son aquellos puntos de los cuales se ha extraído el código correspondiente a los propósitos entrecruzados (crosscutting concerns), donde posteriormente ha de insertarse dicho código con el objeto de seguir manteniendo la semántica del sistema. Algunos ejemplos son las llamadas a métodos, el acceso a atributos o el lanzamiento de una excepción [4].

- **Pointcuts** (puntos de corte): Son un conjunto definido de Join Points, son intersecciones o constructores donde se especifican los puntos de enlace. Por ejemplo, llamadas a los métodos que empiecen por “Set”. Para entender la diferencia entre puntos de enlace y puntos de corte, se puede pensar en estos últimos como las reglas de especificación y en los primeros como las situaciones que satisfacen dichas reglas [4].

- **Advices** (avisos): Este es el código que se debe ejecutar en los puntos de enlace (Joint Points) que se especifiquen en un punto de corte (Point Cut). Los avisos se pueden ejecutar de tres maneras: antes, después o alrededor de los puntos de enlace. Cuando un aspecto se ejecuta “alrededor” quiere decir que se puede, con el aviso, modificar la ejecución del código a la altura del punto de enlace, reemplazarlo o ignorarlo. El cuerpo de un aviso se parece mucho al cuerpo de un método, ya que encapsula la lógica que debe ser ejecutada cuando se alcanza cierto punto de enlace en la ejecución. Los puntos de corte junto con los avisos, son las herramientas

para implementar entrelazado dinámico. Los puntos de corte identifican dónde y los avisos lo completan indicando qué hacer [4].

- **Introduction** (introducciones): Representan el comportamiento que se añade a un objeto en tiempo de ejecución para que además de implementar la interfaz propia implemente una interfaz adicional. Las introducciones se utilizan para el entrelazado estático (Ejemplo AspectJ) [39].

- **Compile-Time declaration** (declaraciones en tiempo de compilación): Esta es otra forma de implementar técnicas de entrelazado estático. Permiten añadir advertencias y errores para que durante la compilación se puedan detectar ciertos patrones de uso que se quiere advertir o prohibir (en el caso de que se identifiquen como errores). Por ejemplo, se puede declarar que es un error hacer uso de las llamadas del paquete java.sql fuera del paquete dataAccess de la aplicación [39].

- **Aspecto:** En el nivel de Diseño**,** un *Aspecto* es un interés que cruza transversalmente clases y componentes jerárquicas.
  En el nivel de implementación, un *Aspecto* es un constructor que nos permite modularizar un interés de corte transversal [1].

- **Weaver** (tejedor): Es el encargado de combinar los Join Points, el comportamiento y los demás elementos de los aspectos a la aplicación. Se encarga de mezclar los diferentes mecanismos de abstracción y composición que aparecen en los lenguajes de aspectos y componentes ayudándose de los puntos de enlace [1].

1.1.2. Etapas en las técnicas de desarrollo de software

- **Primera Etapa: Código Spaghetti**

La figura 1.2 muestra como se mezclan funcionalidad y datos en esta primera etapa. Los problemas [17] que tiene este tipo de desarrollo son los siguientes:

No existe ningún tipo de modelado; al no existir separación entre datos y funciones y al no tener una estructura no es posible aplicar aproximación alguna de modelado, así como tampoco es posible la reutilización de código.

Detección, corrección de errores y mantenimiento son casi imposibles por no existir una clara división entre datos y funciones y porque la estructura de control de flujo es compleja e incomprensible.

Este estilo de programación suele asociarse con lenguajes básicos y antiguos, donde el flujo se controlaba mediante sentencias de control muy primitivas y utilizando números de línea.

Software = Datos (formas) + Funciones (colores)

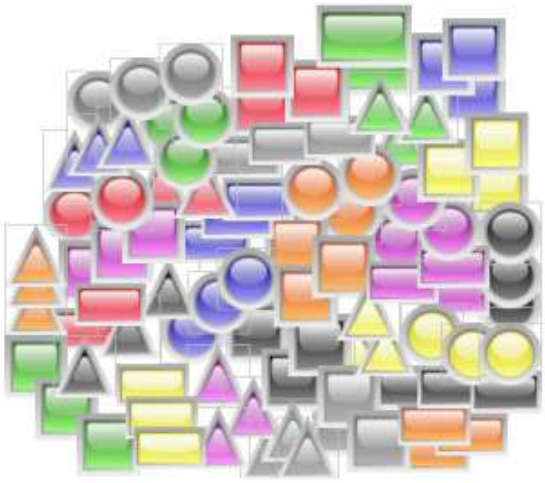

Figura Nº 1.2 Código Spaghetti

- **Segunda Etapa: Descomposición Funcional**

Aparece el concepto de función de tal forma que existe la facilidad para integrar nuevas funciones al código, esto significa que empieza a presentarse por primera vez en el desarrollo de software el concepto de evolución. Sin embargo los datos aún quedan esparcidos por todo el sistema siendo difícil el mantenimiento de los programas, la figura 1.3 muestra como se agrupan los colores que representan las funciones pero los datos (formas) siguen esparcidos, esto hace que no exista la reutilización de código y que los programas sean difíciles de adaptar [17].

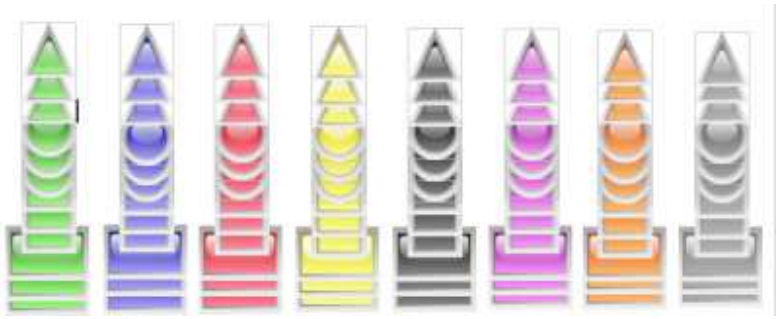

Figura Nº 1.3 Descomposición Funcional

- **Tercera Etapa: Programación Orientada a Objetos (POO)**

POO ha presentado uno de los avances más importantes de los últimos años en la ingeniería de software. Tiene como base el principio de descomposición, debido a que el modelo de objetos utilizado se ajusta mejor a los problemas del mundo real, la figura 1.4 muestra una clara división entre datos y funciones por que se agrupan en objetos; familias de funciones (colores) y familias de datos (formas), nace el concepto de herencia, es decir, unas funcionalidades hijas heredan atributos y operaciones de una funcionalidad padre general.

Además los objetos se pueden modelar (diseñar) con herramientas de ingeniería de software, esto hace que el código sea reusable y que los sistemas implementados con POO sean fáciles de mantener y evolucionar [17].

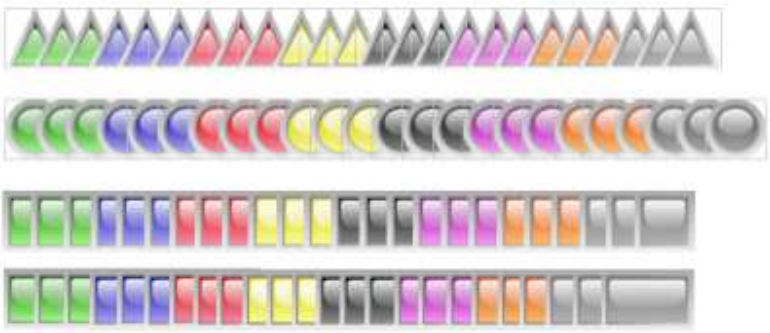

Figura Nº 1.4 Descomposición en Objetos

- **Cuarta Etapa: Programación Orientada a Aspectos**

Esta nueva etapa surge como complemento a la POO, ya que existen funcionalidades que POO no puede encapsular, esta etapa soporta la separación de competencias entre los requisitos funcionales y los no funcionales [17]. En definitiva, lo que persigue es implementar una aplicación de forma eficiente y fácil de entender, la figura 1.5 muestra como además de encapsular en objetos los datos y las funciones AOP separa en aspectos lo que POO no logra encapsular, es decir, los elementos que atraviezan diferentes objetos de la aplicación.

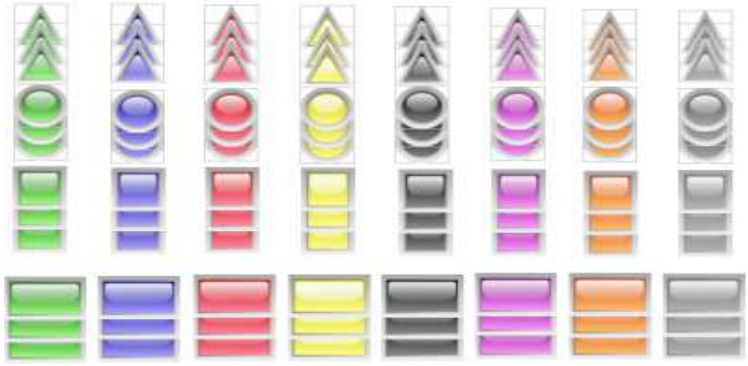

Figura Nº 1.5 Descomposición en Aspectos

1.1.3. Ventajas de AOP

Entre las más importantes se encuentran [17]:

- **Se consigue un código menos enmarañado, más natural y más reducido:** Se logra evitar que las incumbencias que se encuentran dispersas requieran una implementación en varias clases o módulos produciendo un código enmarañado y/o mezclado, el cual será difícil de modificar, entender y reutilizar como se puede ver en la figura 1.6. El servicio de *logging (color rojo)* para Apache Tomcat es requerido en varios puntos de ejecución del sistema que no están relacionados, aparecen responsabilidades atravesando diferentes partes del sistema.

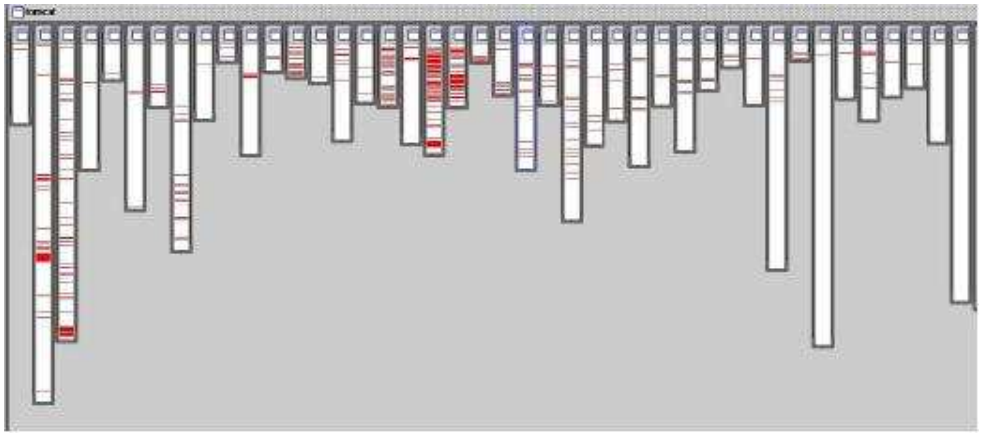

Figura Nº 1.6 *Logging* en Apache Tomcat (Copyright © aspectj.org)

- Mayor facilidad para razonar sobre los conceptos, ya que están separados y las dependencias entre ellos son mínimas. La figura 1.7 muestra como en la implementación sobre Lenguajes de Procedimiento generalizado[3] (LPG), los aspectos se mezclan con la funcionalidad básica, y entre si mismos. En cambio, en la implementación AOP, la separación es completa, con una muy buena *modularización*.

[3] Son los lenguajes orientados a objetos, los procedurales y funcionales.

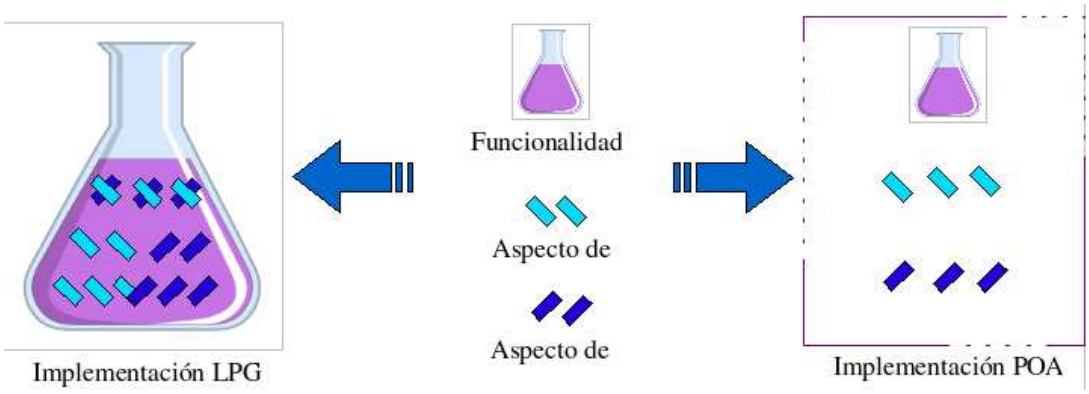

Figura Nº 1.7 Implementación POA Versus LPG

Varias tecnologías con nombres diferentes se encaminan a la consecución de los mismos objetivos y así, el término *Aspect Oriented Programming* o AOP es usado para referirse a varias tecnologías relacionadas como los métodos adaptativos[4], los filtros de composición[5], la programación orientada a sujetos[6] o la separación multidimensional de competencias[7], las cuales son paradigmas de programación cuyo objetivo principal es modularizar desde diferentes perspectivas un proyecto para hacerlo consistente estable y convergente y de esta manera tener un código más reusable.

- **Un código más fácil de depurar y más fácil de mantener:** Se consigue que un conjunto grande de modificaciones en la definición de una materia tenga un impacto mínimo en las otras.

[4] Puede encontrar mayor información en "Demeter Project - Adaptive Programming (AP)". Kart Lieberherr.
[5] Puede encontrar mayor información en "Composition Filtres". Bergmans y Aksit.
[6] Puede encontrar mayor información en "Subject-Oriented Programming – SOP". Harrison y Ossher, IBM.
[7] Puede encontrar mayor información en "Separación de intereses en espacios multidimensionales - Hyperspaces". Tarr y Ossher.

- **Se tiene un código más reusable, que se puede acoplar y desacoplar en cualquier momento:** Permite tener mayor claridad, ver a que clases afecta y a cuales más podría afectar o como podrían tales aspectos afectar a las clases, haciendo los aspectos reutilizables**.**

- **Facilita la creación de documentación y el aprendizaje:** El tener a los aspectos desde todas las etapas permite tener mayor claridad a la hora de tratarlos, ya sea por diseñadores o programadores.

1.1.4. Estructura de un Lenguaje Orientado a Aspectos

En los lenguajes comunes, basta con un compilador o intérprete para traducir un programa a un código entendible por la máquina (Fig. Nº 1.8). En las aplicaciones orientadas a aspectos, además del compilador, se requiere de un tejedor, que combine el código que implementa la funcionalidad básica, con los distintos módulos que implementan los aspectos (Fig. Nº 1.9).

Figura Nº 1.8 Estructura de una implementación en los lenguajes tradicionales.

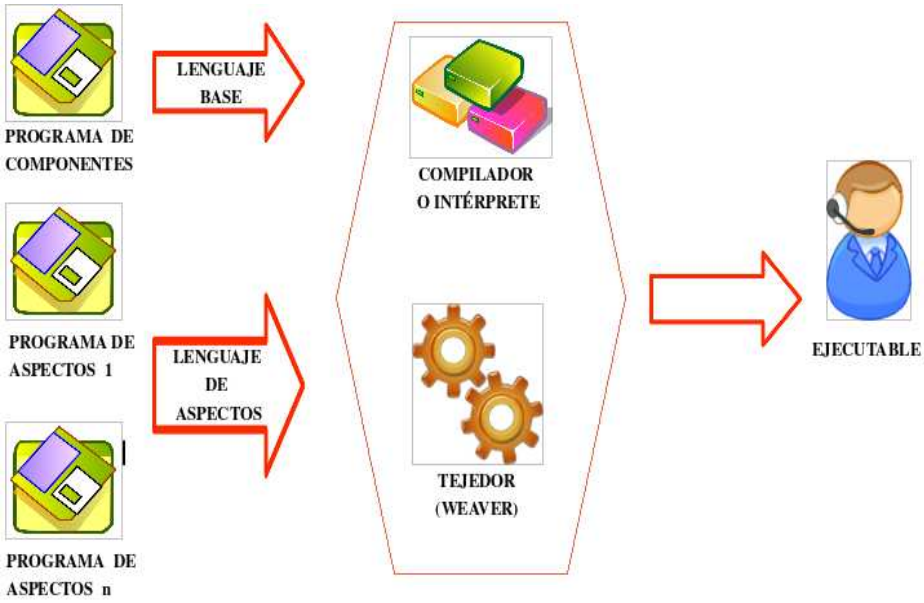

Figura Nº 1.9 Estructura de una implementación en los lenguajes de aspectos.

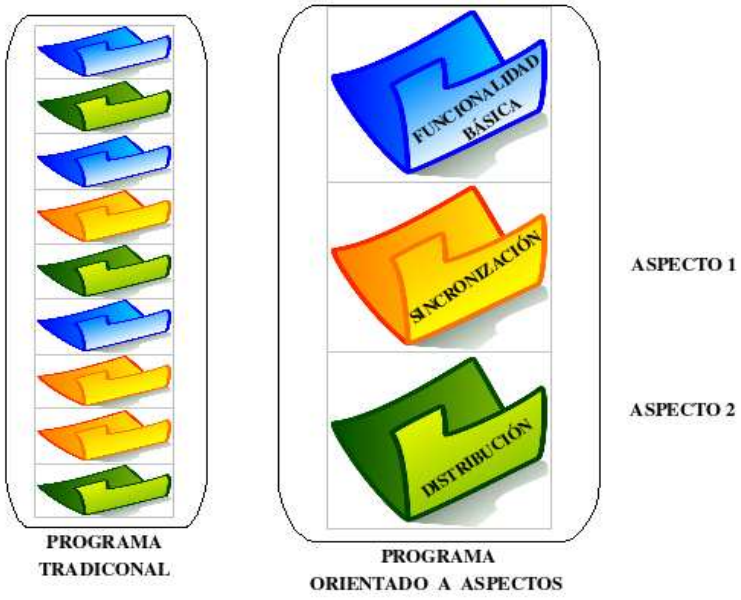

Figura Nº 1.10 Comparación de la forma de un programa tradicional con uno orientado a aspectos

En la figura Nº 1.10 se puede apreciar la estructura de un programa tradicional versus la de un programa con AOP; en el primero se muestran una serie de bloques que representan funcionalidades distintas repartidas por todo el código. En cambio en el programa orientado a aspectos se muestran tres bloques: distribución, sincronización y funcionalidad básica, de esta forma se tiene un programa más compacto y modularizado.

1.1.5 Tejido Estático y Tejido Dinámico.

Existen dos maneras de entrelazar el lenguaje base con el de aspectos:

En el entrelazado estático el código de aspectos se introduce en el código fuente, insertando sentencias en los puntos de enlace. Un ejemplo de este tipo de tejedor, es el tejedor de aspectos de AspectJ [12].

El entrelazado dinámico requiere que los aspectos estén presentes de forma explícita tanto en tiempo de compilación como en tiempo de ejecución [3]. Para esto, tanto los aspectos como las estructuras entrelazadas se deben modelar como objetos y deben mantenerse en el ejecutable. Un tejedor dinámico será capaz de añadir, adaptar y remover aspectos de forma dinámica durante la ejecución. Como ejemplo, el tejedor dinámico AOP/ST [16].

El tejido estático evita la sobrecarga de la aplicación ya que todo el trabajo se realiza en tiempo de compilación. La desventaja es que los aspectos quedan fijos, no pueden ser modificados en tiempo de ejecución, ni existe la posibilidad de agregar o remover nuevos aspectos, la razón es que cuando se realiza la compilación, los aspectos se introducen dentro del código compilado, es decir, se agregan líneas de código y se modifica el programa base o de componentes [4].

El tejido dinámico implica que el proceso de composición se realiza en tiempo de ejecución, decrementando la eficiencia de la aplicación. Esto brinda mayor flexibilidad y libertad al programador, ya que cuenta con la posibilidad de modificar un aspecto según información generada en ejecución, como también introducir o

remover dinámicamente aspectos. El tener que llevar mayor información en ejecución, y tener que considerar más detalles, hace que la implementación de los tejedores dinámicos sea más compleja.

1.1.6. Lenguajes de Aspectos de Propósito General vs. Dominio Específico.

Los *lenguajes de aspectos de dominio específico* [17] solo soportan aspectos para los cuales fueron diseñados. Imponen restricciones en la utilización del lenguaje base. Esto se hace para garantizar que los conceptos del dominio del aspecto se programen utilizando el lenguaje diseñado para este fin y evitar así interferencias entre ambos. Como ejemplos están COOL [2], que trata el aspecto de sincronización, y RIDL [2], para el aspecto de distribución. Los lenguajes de aspectos de dominio específico normalmente tienen un nivel de abstracción mayor que el lenguaje base.

Los *lenguajes de aspectos de propósito general* [17] se diseñaron para ser utilizados con cualquier clase de aspecto. Por lo tanto, no pueden imponer restricciones en el lenguaje base. Normalmente tienen el mismo nivel de abstracción que el lenguaje base y también el mismo conjunto de instrucciones, ya que debería ser posible expresar cualquier código en las unidades de aspectos. Un ejemplo de este tipo de lenguajes es Aspectj [12], que utiliza Java como base, y las instrucciones de los aspectos también se escriben en Java.

1.2. Separación Avanzada de incumbencias (ASoC – *Advanced Separation Of Concerns*)

ASoC es la capacidad de identificar, encapsular y manipular sólo las partes del software que son relevantes a un concepto, meta o propósito en particular. Esto hace posible que se pueda organizar y descomponer el software en partes más pequeñas, manejables y comprensibles para cumplir con requisitos relacionados con la calidad como: adaptabilidad, mantenibilidad, extensibilidad y reusabilidad. Existen dos ideas más poderosas que hay detrás de los aspectos: prescindencia

(*obliviousness*) y cuantificación (*quantification*) [46]. La prescindencia se refiere a que el programador encargado de implementar una funcionalidad específica no debería estar al tanto de las otras dimensiones que pueden afectar su código. La cuantificación es la posibilidad de indicar en qué puntos de unión se aplicará un aspecto, sin necesidad de enumerarlos uno por uno [49].

1.3. Aplicaciones de los Aspectos

Se utilizan los Aspectos para representar características globales que escapan a la ordenación tradicional de capas (presentación, negocio, persistencia, servicios). Se critica AOP diciendo que en realidad no permite hacer nada nuevo que no se pueda hacer con OOP, pero la novedad y la ventaja de AOP no está en el qué, sino en el cómo, pues permite diseños mucho más limpios, aplicaciones mejor construidas y más fáciles de implementar y mantener [13]. Entre las aplicaciones más inmediatas de los aspectos están:

- **Trazas (Logging):** El logging es el registro de la actividad de una aplicación. Se podría asociar determinado código a invocar un método en varios puntos de ejecución, que podrían pertenecer a elementos del programa disconexos (no comunes) y registrar en un archivo de texto o en una base de datos el hecho de que se ha llegado a ese punto de tal forma que el propio método sólo contenga el código para completar la tarea que le corresponde [46].

- **Seguridad:** como se comentó anteriormente, al interceptarse las llamadas a métodos, acceso a miembros (atributos) y demás, se pueden aplicar políticas de seguridad tan arbitrarias como se necesiten sin mezclar ese código con el existente. Esta incumbencia es realmente la autenticación [13].

- **Persistencia:** Con AOP es posible interceptar las llamadas a constructores y mutadores[8] ("setters") de una clase y aprovecharlas para lanzar consultas SQL sincronizando los miembros con la información de la base de datos [13].

- **Trazas de ejecución (Seguimiento o traccing):** Esto es un instrumento de eliminación de fallos muy similar a un logger, pero esto sólo rastrea un tipo de acontecimiento, una ejecución de método [13].

- **Personalización (Profiling)**: Es una incumbencia de eliminación de fallos que mide el tiempo de ejecución consumido en algunos métodos. Esta incumbencia puede ser muy provechosa para descubrir algunos embotellamientos [51].

- **Almacenamiento temporal (Caching):** Con los aspectos se puede aprovechar la intercepción de la llamada a un método para decidir si efectivamente se ejecuta o si se devuelve un resultado almacenado previamente [13]. Caching es la retención de datos, por lo general en uso, busca reducir al mínimo el flujo de tráfico de red y/o accesos de disco.

1.4. Desarrollo de Software Orientado a Aspectos (DSOA – Aspectos en la Etapa de Diseño)

DSOA [42] es una metodología de diseño que tiene como objetivo principal la separación avanzada de incumbencias desde las primeras fases de desarrollo. En un principio la orientación a aspectos se centró en el nivel de implementación y codificación, Actualmente se trabajan muchas propuestas para llevar AOP a nivel de diseño y así facilitar la creación de documentación, el aprendizaje y la reutilización de los aspectos.

La ingeniería de requisitos orientada a aspectos [7] evita los riesgos producidos por los cambios en el ciclo de desarrollo de software a través de la identificación

[8] Los mutadores son métodos especiales que permiten cambiar el valor de los atributos

temprana de aspectos [41]. El término "aspectos tempranos[9] " se refiere a las propiedades transversales en el nivel de requisitos y de arquitectura.

La figura 1.11 muestra la integración de los aspectos a través del ciclo de vida del software.

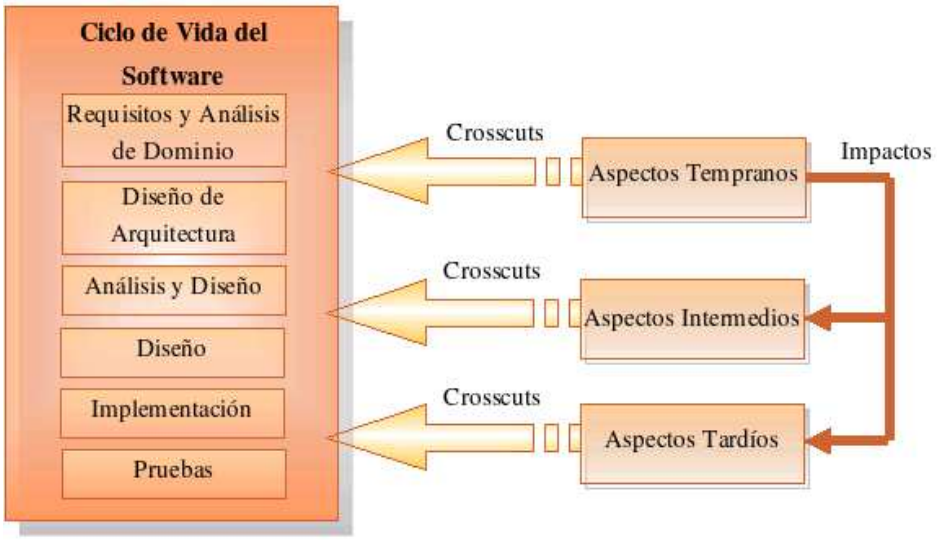

Figura Nº 1.11 Relación entre aspectos tempranos, intermedios y tardíos

1.4.1. DSOA & RUP

El Proceso Unificado de Rational (RUP) promueve una aproximación Iterativa, cada iteración se construye sobre el trabajo de la iteración previa para producir un producto final. DSOA propugna la utilización del concepto de aspectos en todas las fases del ciclo de vida del desarrollo de software. Así, los aspectos aparecen en ingeniería de requisitos, análisis, diseño y en la implementación de las aplicaciones software [42].

[9] Mayor información sobre identificación temprana de aspectos e Ingeniería de Requisitos para DSOA en el sitio oficial. http://www.early-aspects.net/

• Iterativo e incremental

Al igual que en RUP (ver figura 1.12) DSOA puede gestionar un proyecto grande en pequeños proyectos como estrategia para enfrentar el riesgo

• Centrado en la arquitectura

DSOA posee mecanismos para lograr acuerdos de diseño de alto nivel y así garantizar la calidad del producto

• Orientado a casos de uso

DSOA y RUP pueden guiar las tareas del desarrollo alrededor de las necesidades de los clientes.

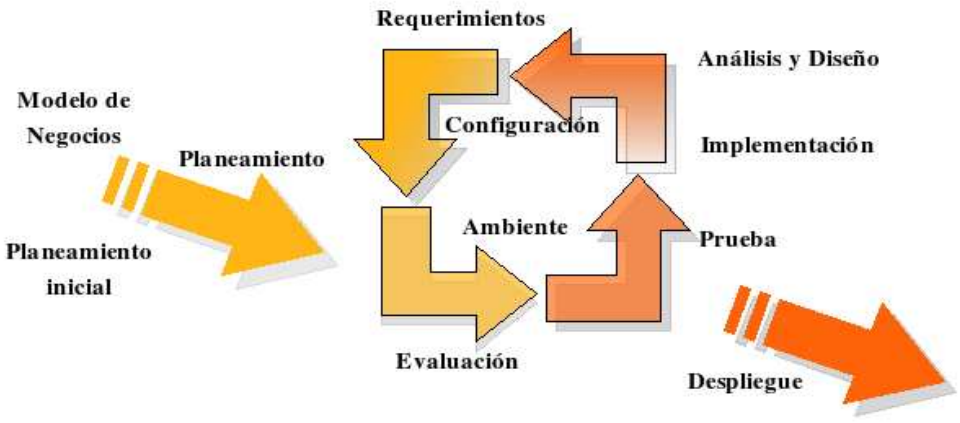

Figura Nº 1.12 Desarrollo Iterativo en RUP (Copyrigth © Introducing the Rational Unified Process)

1.4.2. Separación Avanzada de Incumbencias para Ingeniería de Requisitos

Existe un modelo genérico en [41] para manejar la separación avanzada de incumbencias durante la ingeniería de requisitos (ver figura 1.13); está compuesto de cinco tareas principales:

Tarea 1: Identificar Concerns

A partir de los requisitos se identifican los posibles concerns de un sistema. Para especificar estos requisitos se utilizarán casos de uso, dado que UML [18] puede hoy considerarse un estándar que se puede aplicar casi en cualquier dominio de problema. La tarea de identificación de concerns, está dividida en dos sub-tareas: analizar la lista de casos de uso y analizar información extra provista por el analista. En ambas sub-tareas se genera automáticamente información de interés que se almacena en la tabla 1.1

Incumbencia candidata	Casos de Uso	Actor	Responsabilidad
Nombre de la incumbencia	Lista de casos de uso afectados	Actor relacionado con tal incumbencia	Obligaciones que debe cumplir

Tabla Nº 1.1 Información generada en la tarea 1 en RE

Tarea 2: Elección de aspectos candidatos

Esta tarea se subdivide en dos sub-tareas: Identificar crosscutting concerns y concerns funcionales, y elección de aspectos candidatos por parte del analista.

Tarea 3: Especificar aspectos candidatos

Esta tarea se subdivide en dos sub-tareas: identificar responsabilidades e identificar las relaciones entre aspectos para que los conflictos puedan ser detectados.

	Caso de Uso 1	Caso de Uso 2	....	Caso de Uso N
Aspecto 1				
Aspecto 2	**X**			
….	….	….	….	….
Aspecto N	**X**	**X**		

Tabla Nº 1.2 Relaciones entre Casos de Uso y Aspectos Candidatos

Tarea 4: Identificar conflictos

En esta tarea se identifican todas las posibles situaciones conflictivas entre concerns. Si más de un aspecto es aplicado sobre un mismo caso de uso (ver tabla 1.2), surgirá una situación conflictiva.

Tarea 5: Modelar en UML

Basados en la información obtenida en la realización de las tareas previas, se construirá un modelo visual para el cual se tomará lo mejor de las aproximaciones existentes que soportan aspectos en UML.

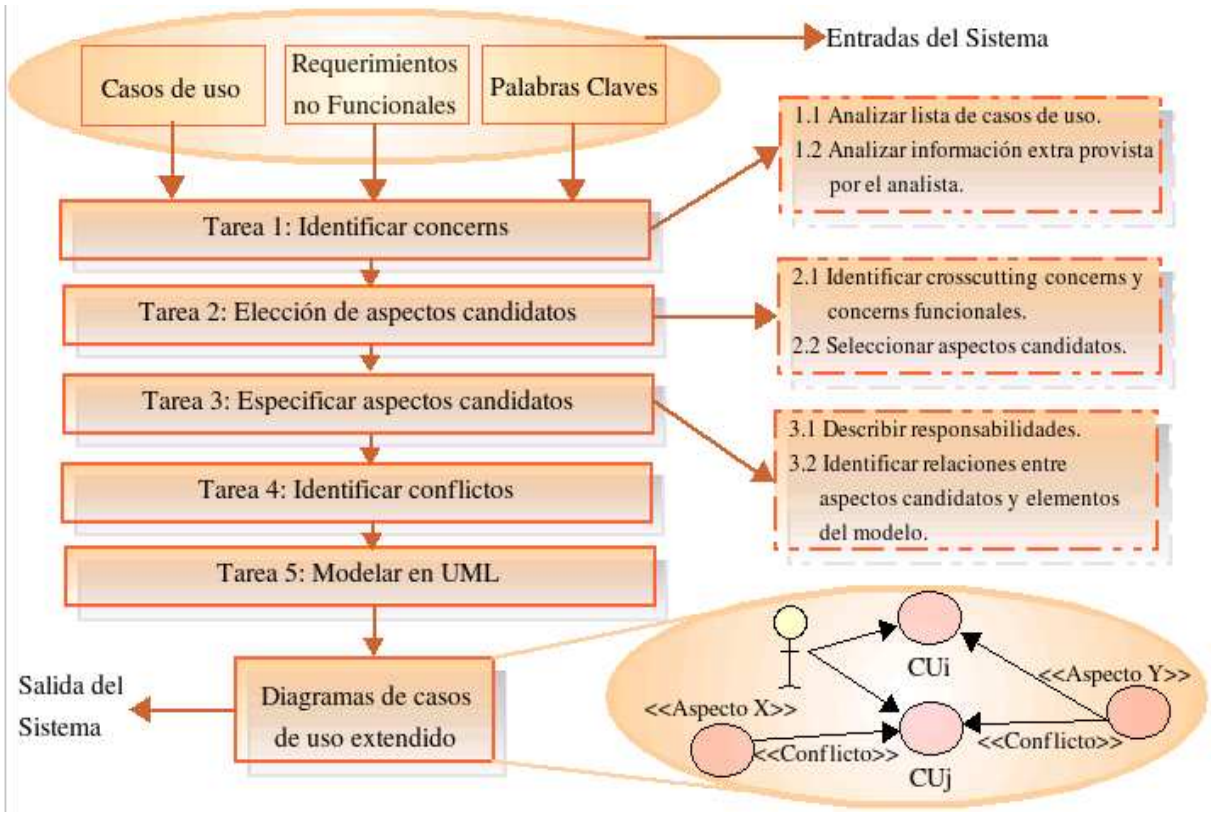

Figura Nº 1.13 Modelo para separación avanzada de incumbencias en Ingeniería de Requisitos (Imagen tomada de [41] bajo licencia de los autores)

1.5. Modelado de aspectos con UML

UML es un lenguaje de modelado extensible [8] que permite el modelado específico de dominio y aumenta la conveniencia como un lenguaje para apoyar el modelado orientado a aspectos; a pesar de que existen muchas aproximaciones en [42] para soportar el modelado de aspectos mediante UML aún no se ha oficializado el uso de UML 2.0 para aspectos.

En estas aproximaciones se pueden distinguir dos líneas generales diferentes. Primero, las extensiones de UML con estereotipos específicos para aspectos concretos; aquí los constructores requeridos por cada aspecto en particular están estereotipados de modo que la herramienta encargada del proceso de enlazado o tejido de aspectos y clases bases pueda determinar los elementos a los que afecta el aspecto en concreto. El otro tipo de aproximaciones opta por una manera más generalizada de soportar los aspectos en UML. Así por ejemplo, se define un metaconstructor, de nombre Aspect, y se definen estereotipos para el comportamiento de los advices o consejos. Las operaciones afectadas por los advices son referenciadas por dichos estereotipos.

Suzuki y Yamamoto proponen en [43] soportar el modelado en el nivel de diseño y enfocarlo hacia la fase de implementación mediante una extensión que soporta aspectos sin romper con la especificación UML existente y con la introducción de una nueva meta clase, en el metamodelo de UML, llamada “aspecto”.

Por otro lado existe una propuesta llamada AML (Aspect Modeling Lenguaje) [34] de Groher y Baumgarth la cual se enfoca desde el punto de vista de paquetes: Un paquete base que contiene la lógica del negocio, un paquete de aspectos que contiene la incumbencia transversal y un conector que se encarga de realizar el enlace entre los aspectos y los elementos base además de encapsular la tecnología de implementación.

Finalmente la propuesta que describe el comportamiento dinámico de los aspectos es la de Basch y Sanchez [44] la cual introduce paquetes de aspectos con Joint

Points, diagramas de clases para aspectos y diagramas de interacción. En el desarrollo de esta propuesta para modelar aspectos en UML, se establecen dos principios: Primero, un aspecto debería ser considerado como un módulo propio encapsulado y separado del sistema, y también separado de otros aspectos. En segundo lugar, se debe determinar que un objetivo clave en el modelado del sistema sería detallar los puntos de enlace (Joint Points) donde los aspectos cortan transversalmente componentes.

1.6. Aplicaciones Web

1.6.1. Definición:

Una aplicación Web es un sistema que los usuarios usan desde un servidor Web a través de Internet o de una intranet. La diferencia entre una página Web y una aplicación Web es que una aplicación Web es un sitio Web donde la navegación a través del sitio, y la entrada de datos por parte de un usuario, afectan el estado de la lógica del negocio. Si no existe lógica del negocio en el servidor, el sistema no puede ser llamado aplicación Web [23][24].

1.6.2. Arquitecturas de aplicaciones Web

Una arquitectura define un conjunto de elementos, conectores, restricciones y un sistema de control que caracterizan a un sistema o a una familia de sistemas [48]. A continuación se nombran algunas arquitecturas existentes para la construcción de aplicaciones Web.

- **Arquitectura de dos capas:** Utilizada en esquemas poco complejos, los datos y los servicios Web aparecen juntos, por tanto es difícil separar los datos de la lógica del negocio.

- **Arquitectura de tres capas:** Los datos y servicios aparecen por separado esto permite mayor facilidad para separar los datos de la lógica del negocio.

1.6.3. Arquitecturas Java para aplicaciones Web

- **Modelo de Arquitectura 1.5 JSP y Servlets**

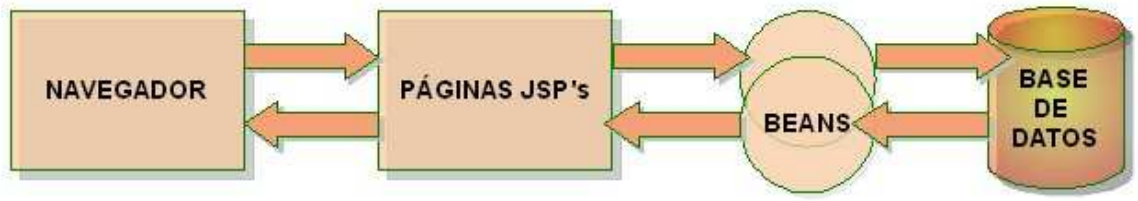

Figura Nº 1.14 Modelo de Arquitectura con JSP's y Servlets

Separación de responsabilidades:

- JSP's llevan la lógica de presentación (navegabilidad, visualización, etc.)
- Beans incrustados asumen las responsabilidades de negocio y datos (ver figura 1.14).

- **Modelo de Arquitectura 2 MVC**

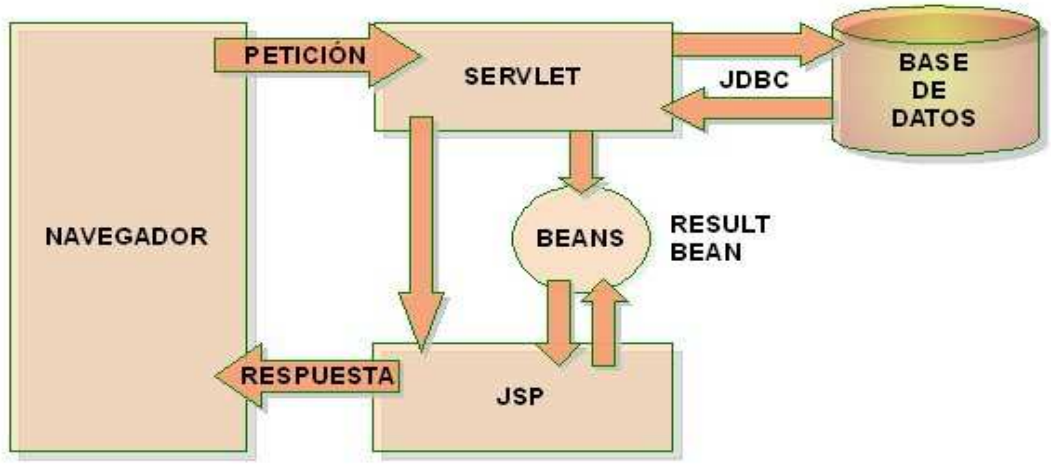

Figura Nº 1.15 Modelo de Arquitectura Java con MVC

Evolución del modelo 1.5

Incorporación del patrón de diseño MVC (ver figura 1.15).

- Controlador: Navegación
- Negocio y Datos: Beans

- Presentación: JSP's

- **Modelo de Arquitectura 2 MVC con Struts**

MVC (Modelo Vista Controlador) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos [46]. De esta manera el diseño queda más simple y permite reutilizar mucho mas código ya que, las partes a implementar, quedan mucho más definidas y son idénticas a las de otros proyectos (ver figura 1.16).

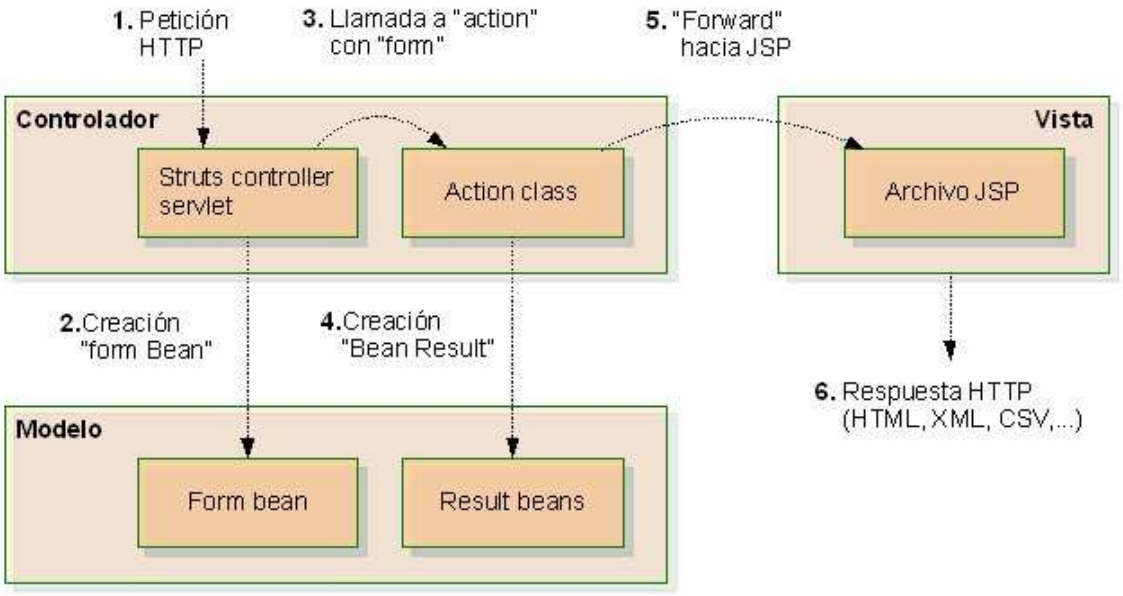

Figura Nº 1.16 Modelo de Arquitectura MVC con Struts

Struts es la implementación del patrón de diseño MVC que aporta Jakarta[10] para aplicaciones Web java ().

[10] Puede encontrar mayor información en el sitio oficial de Jakarta: http://jakarta.apache.org/struts

- **Modelo de Arquitectura 2X Aplicaciones Multicanal**

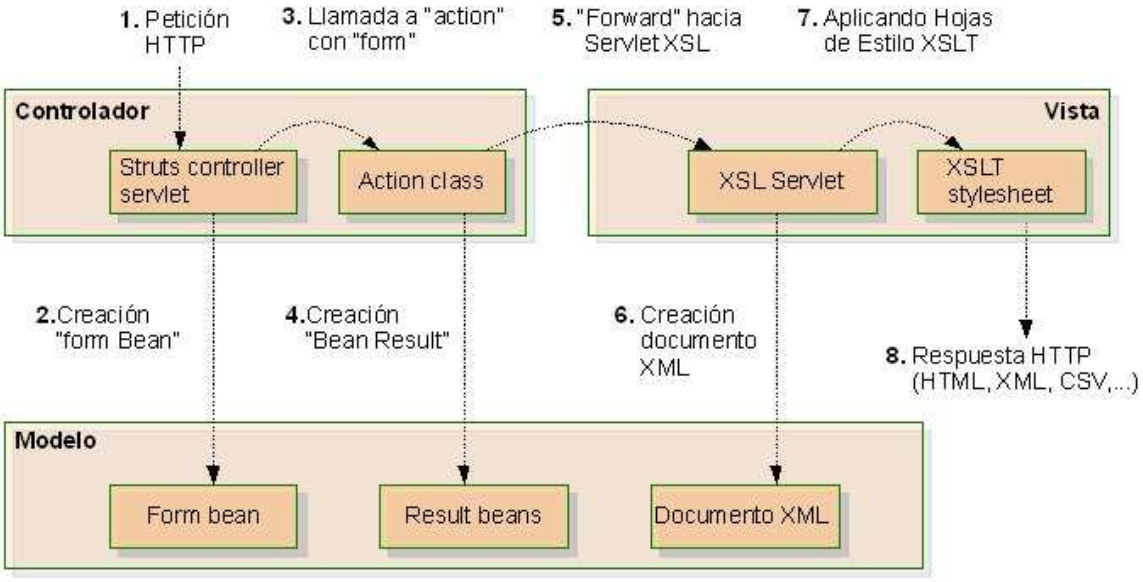

Figura Nº 1.17 Modelo de Arquitectura 2X Aplicaciones Multicanal

Evolución del modelo 2 para construir aplicaciones multicanal.

- Implementación de referencia STXX[11] (extiende Struts)
- Soluciones basadas en XML y XSLT's (ver figura 1.17).

1.6.4. Arquitecturas orientadas a Aspectos

Durante la implementación y mantenimiento los programadores introducen algunas dependencias al código que no encajan con el diseño original de la arquitectura, en [26] se presenta una propuesta basada en algunas aproximaciones que ayudan a que haya correspondencia entre la arquitectura y el código como son: *code inspections, architecture reconstruction, model driven architecture (MDA), enforcement tools* [26].

[11] Puede encontrar mayor información en el sitio oficial: http://stxx.sourceforge.net/

Usando AOP los desarrolladores pueden definir mensajes personalizados de error o advertencia que son impresos cuando el código contiene Joint Points. Estos mensajes personalizados son generados según declaraciones en tiempo de compilación, lo cual es un mecanismo sumamente simple pero poderoso. Las declaraciones en tiempo de compilación funcionan como una verificación adicional en el proceso de construcción, no afectan la compilación porque se pueden agregar o remover en cualquier momento. Este rasgo hace que esta aproximación sea una solución automatizada y un primer paso sin riesgo hacia la adopción de AOP [4].

El software convencional carece de abstracciones para apoyar la representación modular de incumbencias transversales, el trabajo presentado en [23] se enfoca en la representación modular de aspectos a nivel de diseño de arquitecturas de software y provee la intención de integrar las mejores aproximaciones de arquitecturas orientadas a aspectos en un único lenguaje de modelado, para representar aspectos, poincuts, advices, componentes y las relaciones entre estos.

Kandé analiza en su trabajo [24] la conveniencia del uso de UML para el modelado arquitectónico orientado a aspectos, toma un acercamiento que comienza a nivel de código hasta el nivel de descripción de arquitecturas de software, a través del diseño orientado a aspectos usando el estándar UML.

1.7. Herramientas para desarrollo orientado a aspectos

1.7.1. Frameworks Java con Orientación a Aspectos.

A continuación se resumen los frameworks Java existentes más importantes [37]:

- **AspectJ**: Actualmente se encuentra en la versión 1.6.0 [39] [40] Desarrollado inicialmente en Xerox PARC y hospedado actualmente en la fundación eclipse, AspectJ propone una extensión del lenguaje Java, con nuevas palabras

reservadas y construcciones específicas para definir Aspectos, puntos de corte, guías, etc. En AspectJ, un aspecto es una clase, exactamente igual que las clases Java, pero con una particularidad, que pueden contener unos constructores de corte, que no existen en Java. Los cortes de AspectJ capturan colecciones de eventos en la ejecución de un programa. Estos eventos pueden ser invocaciones de métodos, de constructores, lanzamiento de excepciones, acceso a atributos. En AspectJ los aspectos son constructores que trabajan cortando de forma transversal la modularidad de las clases de forma clara y específica.

- **Nanning:** Actualmente en la versión 0.9. Framework basado en proxies dinámicos[12]: clases que se generan dinámicamente en tiempo de ejecución y se interponen entre el autor de una llamada y el código de destino. Los Aspectos se implementan mediante clases Java, y pueden añadirse al sistema programáticamente o declararse en un fichero XML de configuración.

- **AspectWerkz:** Actualmente en la versión 2.0 Al igual que en Nanning, los Aspectos se definen como clases Java y pueden ser añadidos al sistema mediante un fichero de configuración XML, pero además, se pueden declarar los Aspectos y guías en forma de atributos definidos en el código fuente. En la actualidad se utilizan etiquetas xDoclet [38], aunque se prevé adoptar el modelo de metadatos introducido con el JDK 1.5 (JSR 175). AspectWerkz soporta el enlazado de los Aspectos tanto en tiempo de ejecución (mediante proxies dinámicos)[13] como de forma estática, para lo que proporciona un post-procesador que incorpora a las clases compiladas el bytecode correspondiente a la funcionalidad definida en las guías en los puntos de corte seleccionados.

- **JBossAOP:** JBoss AOP [32] es un framework orientado a aspectos 100% Java, fuertemente integrado con el servidor de aplicaciones JBoss. Combinando anotaciones Java *(Java Annotations)* con el JDK 1.5 se consigue

[12] Un Proxy dinámico actúa como un 'cache', es decir un almacenamiento dinámico de información con mayor acceso.

un gran modo de ampliar el lenguaje Java de manera limpia utilizando anotaciones únicamente para la generación de código, además contiene un entorno AOP gracias al cual se pueden proporcionar servicios como gestión de transacciones y persistencia a POJO's ("*Plain Old Java Objects")* de forma transparente. En JBoss AOP se implementan los consejos o advices usando interceptores[14] (*interceptors*). También se pueden implementar las introducciones (*introductions*), las cuales permiten al usuario adicionar métodos o propiedades a una clase existente. Se puede intervenir y 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. Otra función de las introducciones es que permiten implementar herencia múltiple en las clases Java.

Los puntos de corte (*pointcut*) se configuran en un archivo XML externo y contienen información sobre las clases afectadas por los aspectos, los métodos y los parámetros.

Jboss AOP posee un juego de aspectos empaquetados (ver figura 1.18) que son aplicados por medio de anotaciones, expresiones de pointcuts o dinámicamente y en tiempo de ejecución; por lo tanto Jboss AOP trabajo utilizando tejido dinámico.

La figura 1.18 muestra la arquitectura de Jboss AS compuesta de cuatro capas.

[14] Los interceptores son clases java que implementan la interfaz org.jboss.aop y que poseen un consejo o advice con un solo método, mayor información en el sitio oficial de Jboss AOP: http://labs.jboss.com/jbossaop/

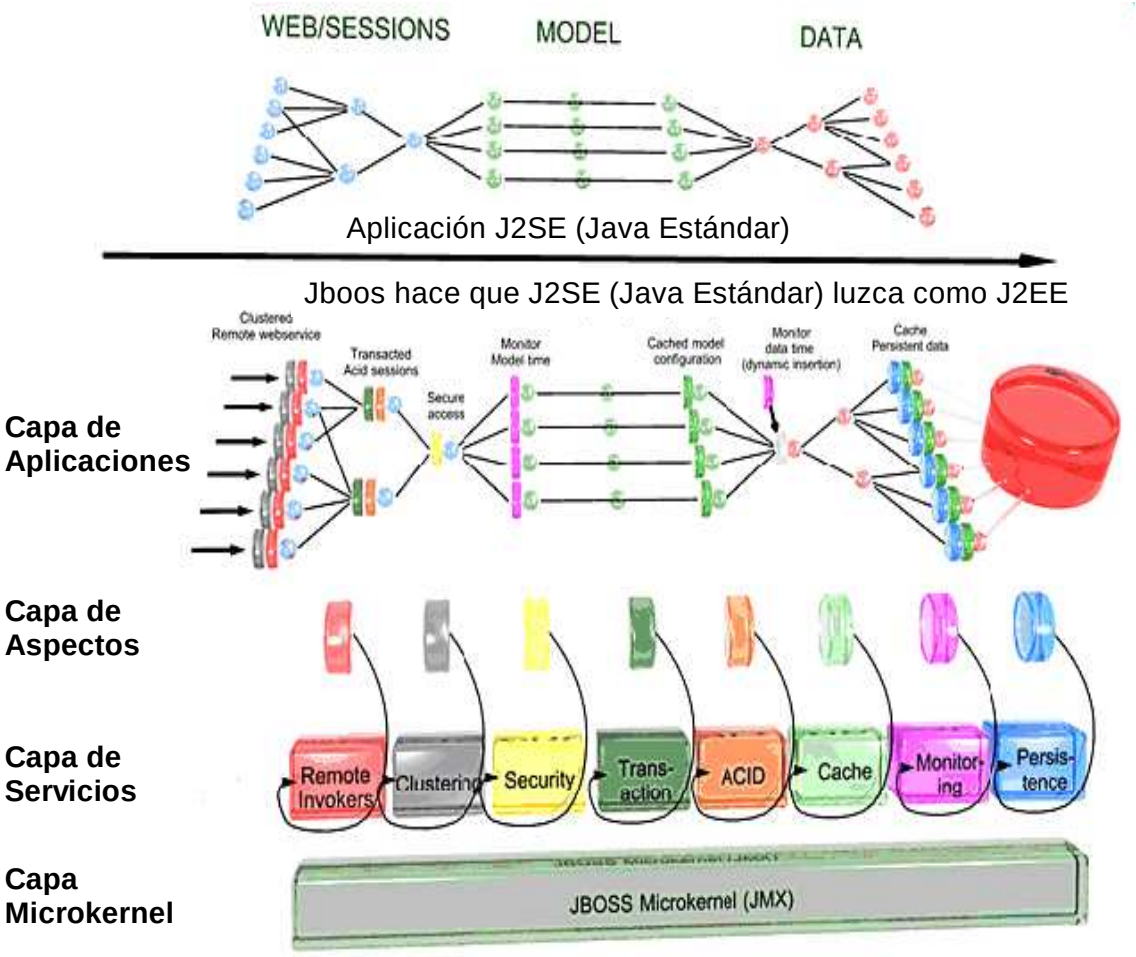

Figura Nº 1.18 Arquitectura de Jboss AS[15]

La capa microkernel es el corazón del servidor y se encarga de la integración de los servicios a través de Java Management Extensions (JMX); la carga de clases y su funcionalidad en tiempo de ejecución.

La capa de Servicios consta de una serie de servicios empaquetados sobre una arquitectura orientada a servicios, (SOA- *Services-Oriented Architecture*). Los servicios de Jboss abarcan la capa de servicios J2EE.

[15] Imagen tomada del sitio oficial de Jboss: http://www.jboss.com/products/jbossas/architecture

La capa de aspectos está basada en el modelo de Programación Orientada a Aspectos (AOP, Aspect-Oriented Programming). Una tradición del Servidor Jboss es el concepto de interceptores. Los interceptores permiten al sistema transparentemente añadir el comportamiento proporcionado por los servicios en cualquier objeto. 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 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 Jboss AS.

- **Spring Framework:** Este framework puede ser utilizado en cualquier servidor J2EE y está dividido en siete módulos (ver figura 1.19).

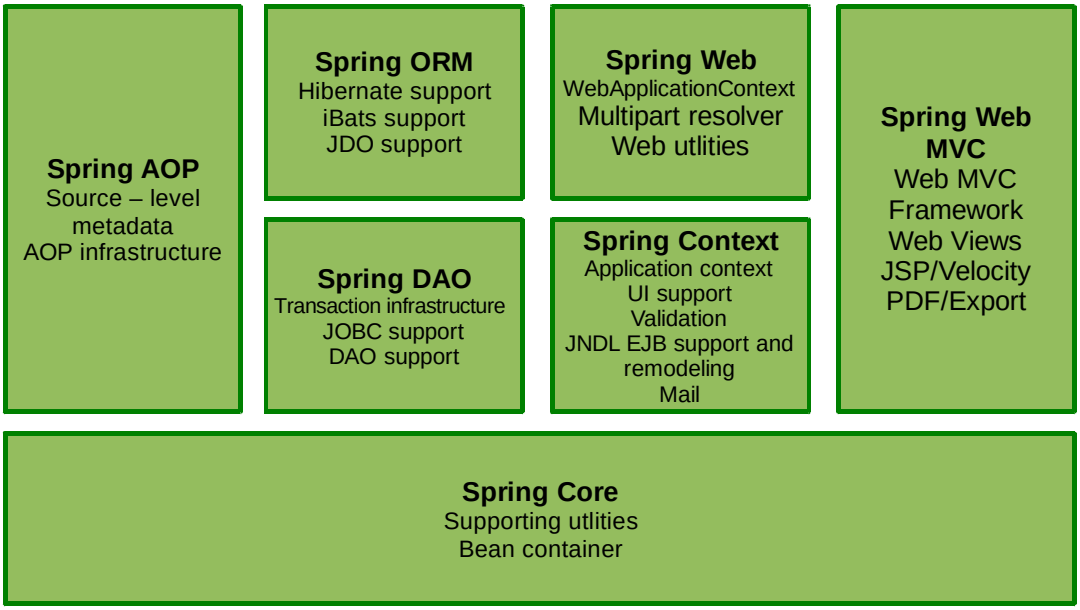

Figura Nº 1.19 Arquitectura de Spring Framework[16]

El módulo AOP integra la funcionalidad de programación orientada a aspectos, permite definir interceptores de métodos y pointcuts al igual que Jboss AOP para

[16] Imagen tomada de http://www.developersbook.com/spring/spring-tutorials/spring-tutorials.php

desacoplar incumbencias transversales. Usando funcionalidades a nivel de meta datos también se puede incorporar en el código todo tipo de información sobre el comportamiento [25].

1.9. Orientación a Aspectos en Aplicaciones Web con lenguajes de Script

Los lenguajes de script son lenguajes para ser interpretados, no para ser compilados, por lo que no generan ningún archivo ejecutable. Entre los más conocidos se encuentran Javascript, VBscript, PHP, PERL, TCL/TK entre otros [14]; todos estos lenguajes se representaban a duras penas por medio de clases en un principio pero hoy en día empiezan a introducir conceptos de AOP [13] [15].

1.9.1. PHP orientado a aspectos

La ventaja de utilizar un lenguaje de script como PHP a la hora de realizar servicios o aplicaciones Web es que es un lenguaje fácil de implementar por su similitud con C y PERL, además de ser gratuito es multiplataforma; funciona en todas las plataformas que soporten apache. Existe una comunidad a nivel mundial que realiza aportes a diario; por ejemplo librerías de funciones que soportan casi cualquier labor: encriptación, envió de correo, gestión de comercio electrónico, XML, creación de PDF y acceso a datos, entre otras. Por estas razones PHP es una de las herramientas más utilizadas en aplicaciones Web [50].

Hoy en día para desarrollar aplicaciones Web se requiere de tecnologías que sean capaces de separar lógica presentación y contenido tal y como lo hace PHPaspect basándose en XML y XSLT[17].

A continuación se nombran algunas herramientas PHP que soportan orientación a aspectos:

[17] Para mayor información visitar el sitio oficial XSL Transformations (XSLT). Oficial W3C Recommendation, http://www.w3.org/TR/xslt

- **PHP orientado a aspectos (AoPHP)**

  AoPHP [20] es una adición a PHP que permite orientación a aspectos en middleware [21]. AoPHP fue escrito y desarrollado para el empleo en aplicaciones Web grandes. AoPHP puede ser un instrumento muy poderoso cuando es usado correctamente en el desarrollo Web. Con muchos empleos posibles en líneas de aplicaciones de hoy en día, sabiendo usar AoPHP con eficacia puede ser muy práctico. Actualmente es desarrollado en Java 1.5 como un preprocesador a PHP. Sin embargo las versiones futuras serán escritas en alguna variante de C como un módulo apache.

- **MFAOPHP**

  Es una implementación simple de AOP en PHP, esta soporta JoinPoints, PointCuts; conceptos de AOP mencionados anteriormente, antes, después y alrededor de los aspectos. No tiene preprocesador PHP, pero teje Aspectos en tiempo real [19].

- **PHPaspect**

Es una extensión de PHP la cual implementa Programación Orientada a Aspectos, el compilador phpAspect[18] teje aspectos implementando incumbencias transversales en el código PHP. El proceso de tejido es estático (antes de la ejecución del código) y basado en análisis Lex y Yacc [22] para generar análisis de archivos XML (XML Parse trees). Para realizar la transformación de código original sobre aquellos árboles se utiliza XSLT. El proceso de tejido entre los aspectos y el código PHP puede ser ejecutado con PHP 5.0 o superior.

PhpAspect contiene todos los joinpoints tradicionales. Sin embargo, el objetivo de esta herramienta es ofrecer una AOP que contenga todas las incumbencias específicas para el desarrollo Web en PHP. La figura 1.20 muestra la arquitectura de phpAspect.

[18] Para mayor información visitar el sitio oficial phpAspect http://phpaspect.org

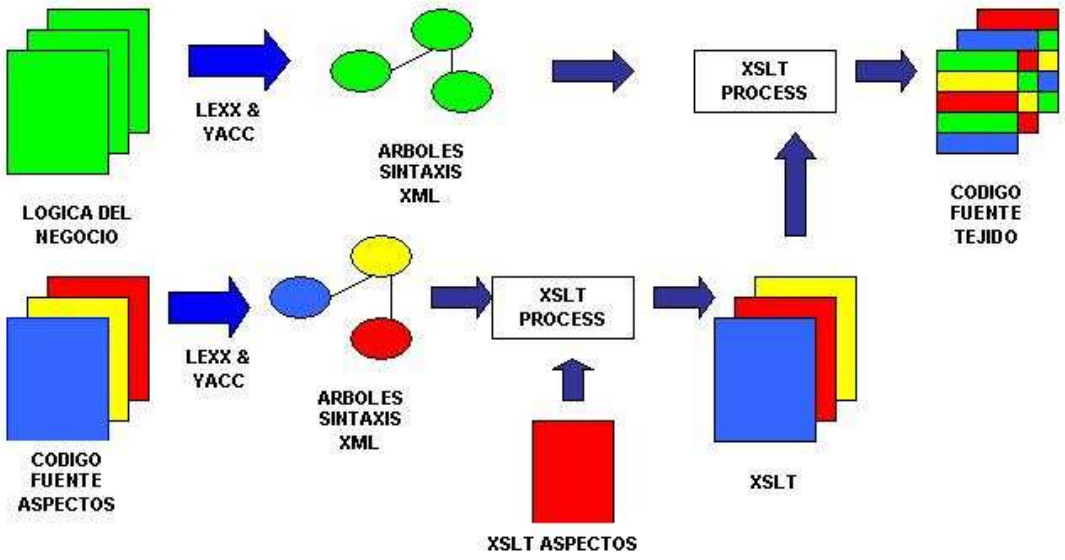

Figura Nº 1.20 Arquitectura PhpAspect[19]

1.10. Comparación entre las distintas herramientas para desarrollo con AOP

A continuación se detalla una tabla comparando las características que implementa cada herramienta con respecto a la AOP.

Lenguaje OA	Lenguaje Base	Tejido	Herramienta Desarrollo	Licencia	Características
AspectJ	Java	Estático	IDE para Eclipse, Jbuilder y Emacs	Abierta bajo Mozilla Public License	cambia el .class de la clase para añadirle los aspectos, nueva sintaxis
JBoss AOP	Java	Dinámico	Servidor de Aplicaciones J2EE	Licencia bajo LGPL	desarrollado por JBoss Integrado al servidor de aplicaciones
JAC	Java	Dinámico	UML IDE con soporte a aspectos	Licencia bajo LGPL	JAC (Java Aspect Components) framework para

[19] Imagen tomada del sitio oficial de phpAspect http://www.phpaspect.org/

					aplicaciones distribuidas OA en Java
Spring AOP	Java	Dinámico	Framework	Licencia bajo LGPL	Utiliza proxys dinámicos de manera que basta con la JDK para funcionar
COOL	Cualquier lenguaje OO	Estático	Framework	Lopes, C.I.V. *D: A Language Framework for Distributed Programming*. Phd. Thesis.	Describe la Sincronización de hilos concurrentes. Visibilidad limitada del aspecto.
RIDL	Cualquier lenguaje OO	Estático	Framework	Lopes, C.I.V. *D: A Language Framework for Distributed Programming*. Phd. Thesis.	Describe la distribución Visibilidad limitada del aspecto.
AspectC++	C++	Estático	Plugin Eclipse ACDT	Opensource bajo EPL (Eclipse Public License)	Aspectos son Extensiones del concepto de clase.
PHPAspect	PHP	Estático	Funciona con librerías adicionales PEAR, próximamente plugin para eclipse APDT	Licencia bajo Creative Commons Attribution License	Genera análisis de archivos XML (árboles XML). Para la transformación de código se utiliza XSLT

Tabla Nº 1.3 comparación entre las distintas herramientas para desarrollo con AOP

II. ARQUITECTURA BASADA EN AOP PARA EL DESARROLLO DE APLICACIONES WEB

2.1. Descripción General

"Una arquitectura de software es una descripción de subsistemas y componentes de un sistema software y las relaciones entre estos".[20]

La arquitectura orientada a aspectos está desarrollada utilizando la metodología DSOA; como primera instancia (requisitos) se utiliza la aproximación "identificación temprana de aspectos" [41]. Esta aproximación permite identificar los aspectos más importantes que se encuentran dispersos por una aplicación Web, se trata de cubrir la mayoría de servicios que están presentes en una aplicación Web por medio de casos uso para de esta forma identificar que aspectos atraviesan dichos casos de uso y así decidir que incumbencias son las más importantes para diseñar la arquitectura.

Cada incumbencia transversal identificada se describirá modelando cada aspecto por separado por medio de la aproximación AML [34], la cual fue descrita en el capítulo anterior y al final se obtendrá como resultado una capa de aspectos que puede ser insertada en cualquier modelo arquitectónico multicapa.

2.2. Proceso de Desarrollo

El desarrollo de la arquitectura cubrió dos etapas: análisis y diseño, en las cuales se aplica la metodología "identificación temprana de aspectos" adaptada a las dos primeras etapas del RUP[21]. La etapa de análisis cobró una importancia adicional debido al hecho de que se trata de una arquitectura que sirve como base para desarrollar aplicaciones Web orientadas a aspectos, por tal razón dentro del

[20] Definición tomada del libro: "Wiley-Pattern-Oriented Software Architecture – A System of Patterns, Volume I"
[21] Proceso Unificado de Rational

diseño arquitectónico, lo más importante fuera de la identificación de aspectos en una aplicación Web es su especificación e integración con otros componentes arquitectónicos para que la arquitectura pueda ser aplicada.

Basado en los trabajos existentes, se propone una aproximación aspectual en el nivel lógico de diseño arquitectónico para el desarrollo de aplicaciones Web.

A continuación se muestran los resultados más importantes en cada etapa de desarrollo:

2.2.1. Análisis

El problema de las actuales arquitecturas Web es que no consideran las incumbencias transversales que atraviesan distintos elementos, afectando componentes y conectores [48]. El aporte del presente trabajo es definir una arquitectura no formal para aplicaciones Web, la cual identifique algunas incumbencias transversales y las encapsule utilizando AOP.

Para representar los elementos de la arquitectura se debería utilizar un ADL (lenguaje de descripción de arquitecturas) [48], sin embargo los ADL's son lenguajes complicados de adoptar, solo se especializan en un tipo de sistemas y no existe un ADL formal para soportar aspectos; por esto se aprovecha el hecho de que DSOA utiliza las extensiones UML para dar soporte al modelado de aspectos en un nivel de abstracción alto (nivel arquitectónico); aunque no se podría considerar UML 2.0 como un ADL, algunas investigaciones como la de Medvidovic [48] han probado que UML puede utilizarse como metalenguaje para simular otros ADL's, por otra parte en [35] se presenta una aproximación para modelar aspectos a nivel arquitectónico por medio de conectores e interfaces ya que para realizar una descripción arquitectónica se tienen en cuenta dos elementos: composición e interacción que corresponden de manera general a las perspectivas estática y dinámica en UML. La Arquitectura de Software es, ante todo, un modelo descriptivo basado en la especificación de composición e interacción [48].

DSOA propone considerar durante la arquitectura del software los aspectos como entidades de primera clase (aspectos arquitectónicos) [36] que describen el comportamiento del sistema. Sin embargo, los nuevos elementos introducidos en el sistema tienen que integrarse en él. Para ello, deben especificarse los nuevos aspectos como componentes de diseño y definirse para ellos sus relaciones con el sistema base y así poder integrar aspectos y componentes en una arquitectura final.

A partir de los trabajos existentes y de las aproximaciones para soportar aspectos (nivel requisitos) se utilizan las dos primeras etapas de la metodología RUP (análisis y diseño) ya que como se describió en la sección 1.4.1 DSOA posee características similares a RUP, por lo tanto se puede adaptar a sus dos primeras etapas.

Para aplicar la metodología "identificación temprana de aspectos" [41] los casos de uso del análisis, los cuales fueron escogidos como casos de uso claves en cualquier aplicación Web serán las entradas (primera tarea) de dicha metodología; en la parte de diseño (última tarea) se modelarán los aspectos que van a conformar una capa para la arquitectura. Para modelar dichos aspectos se utiliza un perfil UML adecuado; en este caso AML.

Producto del análisis para una aplicación Web convencional se identificaron dos actores principales interactuando con el sistema:

Administrador: Es la persona encargada de gestionar todo el contenido de una aplicación Web.

Usuario: Es la persona que va a acceder y va a hacer uso de los servicios de la aplicación Web.

A continuación se describen los requisitos más comunes de manera global que debe cumplir una aplicación Web convencional, es decir, primero se modelarán los elementos de la lógica del negocio de la manera más general posible utilizando los mecanismos UML disponibles para su posterior reutilización e integración con los aspectos.

- Login: Se refiere a ingresar a la aplicación con una cuenta y unos privilegios previamente establecidos o ingresar a una parte restringida de la aplicación Web (zona de administración).

- Ver Contenido: Se refiere a un servicio que muestre los contenidos de una aplicación Web dependiendo de la solicitud del usuario.

- Buscar Contenido: Realizar búsquedas de palabras claves o archivos dentro de la aplicación.

- Cargar contenido: capacidad para añadir contenido que puede ser insertado dentro de una base de datos dependiendo de la aplicación Web.

- Descargar Contenido: Hace relación a poder bajar archivos de diferente tipo desde la aplicación Web (texto, documentos, fotografías, PDF's, etc.), esto dependiendo de unos permisos asignados.

- Crear usuario: Se refiere a la capacidad de poder registrarse y crear su propio perfil, también hace relación a que un administrador pueda agregar usuarios a la aplicación y les asigne permisos a los mismos.

- Eliminar Usuario: Capacidad para borrar un usuario registrado.

- Mostrar Usuario: Este caso de uso permite poder ver todos los usuarios que se encuentran registrados.

- Enviar Correo: Capacidad para poder efectuar el envio de correo electrónico.

- Efectuar transacción: se refiere a efectuar transacciones bancarias a través de Internet.

La figura Nº 2.1 muestra el diagrama de casos de uso del análisis obtenido a partir de los requisitos generales.

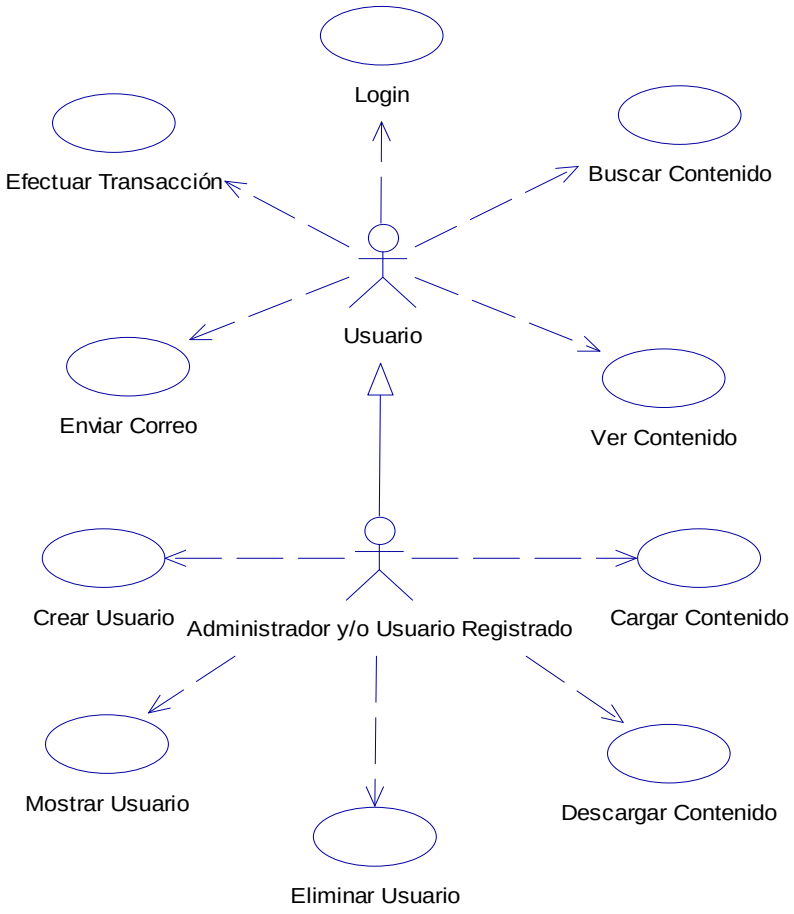

Figura Nº 2.1 Diagrama de Casos de Uso del Análisis[22]

[22] Escogidos de acuerdo a la intención de poder abarcar los servicios más importantes presentes en una aplicación Web

Una vez identificados algunos de los requisitos generales más importantes de una aplicación Web (sistema de componentes) se identifican las funcionalidades que atraviesan o están esparcidas a través de dichos componentes para diseñar la arquitectura.

2.2.2. Aplicación de la Metodología DSOA

Entradas del sistema (Especificación de los casos de uso)

Listado de Casos de Uso:

1. Login (Ver tabla 2.1)
2. Ver Contenido (Ver tabla 2.2)
3. Buscar Contenido (Ver tabla 2.3)
4. Cargar Contenido (Ver tabla 2.4)
5. Descargar Contenido (Ver tabla 2.5)
6. Crear Usuario (Ver tabla 2.6)
7. Eliminar Usuario (Ver tabla 2.7)
8. Mostrar Usuario (Ver tabla 2.8)
9. Enviar Correo (Ver tabla 2.9)
10. Efectuar transacción (Ver tabla 2.10)

Caso de Uso	**Login**	
Descripción	Permite a un usuario registrado y/o administrador ingresar a cierto sitio de la aplicación Web, previa verificación de login y password.	
Actor	Usuario, Administrador	
Trigger	El usuario selecciona la opción ingresar	
Flujo de Eventos	**Flujo Básico**	El sistema verifica que el login y password sean correctos
		El sistema permite el acceso al usuario mostrando el sitio restringido
	Flujo Alternativo	Si los datos de validación ingresados por el usuario son incorrectos el sistema muestra un mensaje de error y se reinicia el caso de Uso

Requisitos especiales	Conexión segura a la base de datos de usuarios.	
Suposición	Debe existir un registro de login y password en la base de datos	
Post_Condiciones		
Puntos de extensión		

Tabla Nº 2.1 Especificación del Caso de Uso Login

Caso de Uso	**Ver Contenido**	
Descripción	Permite a un usuario cualquiera a través de una consulta o selección de un menú ver cualquier tipo de contenido.	
Actor	Administrador y/o usuario avanzado	
Trigger	El usuario selecciona la opción gestionar usuarios.	
Flujo de Eventos	**Flujo Básico**	El sistema verifica que el usuario este autorizado para realizar la consulta o para acceder a cierta opción de menú.
		El sistema despliega la información solicitada por el usuario
	Flujo Alternativo	Si el usuario no esta autorizado para acceder al servicio el sistema despliega un mensaje de error y se reinicia el caso de Uso.
Requisitos especiales	Conexión segura a la base de datos de contenidos y usuarios.	
Suposición	El usuario debe tener los permisos para acceder a la consulta	
Post_Condiciones		
Puntos de extensión	El sistema utiliza los casos de uso buscar contenido y cargar contenido.	

Tabla Nº 2.2 Especificación del Caso de Uso Ver Contenido

Caso de Uso	**Buscar Contenido**	
Descripción	Permite a un usuario buscar contenidos dentro de la aplicación Web.	
Actor	Usuario	
Trigger	El usuario selecciona la opción buscar	
	Flujo Básico	El sistema busca los datos solicitados dentro de los contenidos del sitio Web.

	Flujo Alternativo	Si los datos de búsqueda no son válidos o no se encuentran el sistema despliega un mensaje respectivo.
Requisitos especiales		
Suposición	Los datos buscados deben encontrarse dentro del contenido.	
Post_Condiciones		
Puntos de extensión	El sistema utiliza el caso de uso Ver contenidos.	

Tabla Nº 2.3 Especificación del Caso de Uso Buscar Contenido

Caso de Uso	**Cargar Contenido**	
Descripción	Permite a un administrador o usuario avanzado subir cualquier tipo de contenidos a la aplicación Web, por ejemplo, texto, documentos, PDF's, imágenes, etc.	
Actor	Administrador	
Trigger	El usuario selecciona Cargar Contenido	
Flujo de Eventos	**Flujo Básico**	El sistema verifica que el administrador este autorizado para realizar tal operación
		El sistema despliega un menú de opciones para subir los contenidos.
		El sistema almacena la información de contenidos.
	Flujo Alternativo	Si los datos ingresados no son válidos el sistema muestra un mensaje de error y se reinicia el caso de Uso.
Requisitos especiales	Conexión segura a la base de datos de contenidos.	
Suposición		
Post_Condiciones		
Puntos de extensión	La confirmación de este caso de uso se realiza cuando se lanza el caso de uso Ver contenidos	

Tabla Nº 2.4 Especificación del Caso de Uso Cargar Contenido

Caso de Uso	**Descargar Contenido**
Descripción	Permite a un administrador o usuario autorizado descargar cualquier tipo de contenidos a la aplicación Web, por ejemplo, texto, documentos, PDF's, imágenes, etc.

Actor	Administrador y/o usuario avanzado	
Trigger	El usuario selecciona Descargar Contenido	
Flujo de Eventos	**Flujo Básico**	El sistema despliega un menú de opciones para descargar los contenidos.
		El sistema almacena un registro de la operación.
	Flujo Alternativo	Si los datos solicitados para la descarga no se encuentran disponibles, el sistema muestra un mensaje de error y se reinicia el caso de Uso.
Requisitos especiales	Conexión segura a la base de datos de contenido.	
Suposición	El usuario debe tener los privilegios para realizar la descarga de contenido.	
Post_Condiciones		
Puntos de extensión		

Tabla Nº 2.5 Especificación del Caso de Uso Descargar Contenido

Caso de Uso	**Crear Usuario**	
Descripción	Permite a un administrador crear un usuario con un perfil y asignar permisos para ingreso a la aplicación.	
Actor	Administrador y/o usuario avanzado	
Trigger	El administrador selecciona la opción gestionar usuarios.	
Flujo de Eventos	**Flujo Básico**	El sistema verifica que el administrador este autorizado para realizar tal operación
		El sistema despliega un formulario de registro y un menú de opciones para fijar las características del usuario.
		El sistema almacena la información del usuario.
	Flujo Alternativo	Si los datos ingresados no son válidos el sistema muestra un mensaje de error y se reinicia el caso de Uso.
Requisitos especiales	Conexión segura a la base de datos de usuarios.	
Suposición	El administrador debe haber ingresado a la zona de administración, previa autenticación.	
Post_Condiciones		

Puntos de extensión	El sistema confirma los resultados obtenidos mediante el caso de uso Mostrar Usuarios

Tabla Nº 2.6 Especificación del Caso de Uso Crear Usuario

Caso de Uso	**Eliminar Usuario**	
Descripción	Permite a un administrador eliminar un usuario.	
Actor	Administrador y/o usuario avanzado	
Trigger	El administrador selecciona la opción Eliminar Usuario.	
Flujo de Eventos	**Flujo Básico**	El sistema verifica que el administrador este autorizado para realizar tal operación
		El sistema almacena despliega un mensaje de confirmación
	Flujo Alternativo	Si no se puede eliminar el usuario por falla en la conexión a la base de datos el sistema muestra un mensaje de error.
Requisitos especiales	Conexión segura a la base de datos de usuarios.	
Suposición	El administrador debe haber ingresado a la zona de administración, previa autenticación.	
Post_Condiciones		
Puntos de extensión	El sistema confirma los resultados obtenidos mediante el caso de uso Mostrar Usuario	

Tabla Nº 2.7 Especificación del Caso de Uso Eliminar Usuario

Caso de Uso	**Mostrar Usuario**	
Descripción	Permite a un administrador ver el perfil de un usuario.	
Actor	Administrador y/o usuario avanzado	
Trigger	El administrador selecciona la opción Mostrar usuarios.	
Flujo de Eventos	**Flujo Básico**	El sistema verifica que el administrador este autorizado para realizar tal operación
		El sistema despliega los datos del usuario respectivo
	Flujo Alternativo	Si no se puede establecer conexión con la base de datos el sistema despliega un mensaje de error.
Requisitos especiales	Conexión segura a la base de datos de usuarios.	
Suposición	El administrador debe haber ingresado a la zona de administración, previa autenticación.	

Post_Condiciones	
Puntos de extensión	El sistema confirma los resultados obtenidos mediante el caso de uso Mostrar Usuarios

Tabla Nº 2.8 Especificación del Caso de Uso Mostrar Usuario

Caso de Uso	**Enviar Correo**	
Descripción	Permite a un usuario registrado poder efectuar el envío de correo electrónico.	
Actor	Administrador y/o usuario avanzado	
Trigger	El administrador selecciona la opción gestionar usuarios.	
Flujo de Eventos	**Flujo Básico**	El sistema verifica que el usuario este registrado y este autorizado para realizar tal operación
		El sistema despliega un menú con las opciones de la cuenta de correo del usuario.
		Una vez se realice el envío de correo, el sistema muestra un mensaje de confirmación.
	Flujo Alternativo	Si los datos ingresados no son válidos (dirección electrónica destinatario, cuerpo del texto, archivos adjuntos, etc.) el sistema muestra un mensaje de error y se reinicia el caso de Uso.
Requisitos especiales	Conexión segura a la base de datos de correo electrónico.	
Suposición	El usuario debe haber ingresado a su cuenta de correo, previa autenticación.	
Post_Condiciones		
Puntos de extensión		

Tabla Nº 2.9 Especificación del Caso de Uso Enviar Correo

Caso de Uso	**Efectuar Transacción**	
Descripción	Permite a un usuario cualquiera realizar una transacción bancaria a través de un portal Web.	
Actor	Usuario	
Trigger	El usuario selecciona la opción efectuar transacción	
	Flujo Básico	El sistema verifica que el usuario este autorizado para realizar la

		operación
		El sistema despliega las opciones de Efectuar Transacción.
		El sistema muestra un mensaje de confirmación una vez terminada la transacción y almacena datos actualizados.
	Flujo Alternativo	Si la transacción no fue realizada exitosamente el sistema muestra un mensaje de error.
Requisitos especiales	Conexión segura a la Base de Datos	
Suposición	El usuario debe encontrarse dentro de la base de datos y cumplir con ciertos requisitos, por ejemplo tener crédito suficiente para realizar la transacción	
Post_Condiciones		
Puntos de extensión		

Tabla Nº 2.10 Especificación del Caso de Uso Efectuar Transacción

2.2.3. Requisitos no funcionales del sistema

Palabras Claves del Sistema: las palabras claves se generan a partir de los requisitos no funcionales más relevantes del sistema, estos requisitos son aquellos relacionados con las propiedades del sistema que tienen que ver con rendimiento, velocidad, uso de memoria, tiempo de respuesta media; puede que no pertenezcan a ningún caso de uso y se agregan como requisitos adicionales [27]. Entre los identificados para una aplicación Web se encuentran: seguridad, persistencia, sincronización, navegabilidad, disponibilidad.

2.2.4. Tarea 1. Identificar Concerns

Sub-Tarea 1.1: Analizar lista de Casos de Uso

Los verbos relevantes encontrados a partir de las especificaciones de los casos de uso son los siguientes: verificar, desplegar, mostrar, conectar, accesar, almacenar. Estos verbos se considera podrían ser los concerns o incumbencias iniciales.

Sub-Tarea 1.2: Analizar información extra provista por el analista

De las palabras claves que se encuentran en los requisitos funcionales de los casos de uso provistos por el analista, en este caso provistos por los casos de uso del análisis se encuentra repetida la palabra “verifica” y “seguro” en la descripción de los casos de uso “Login” (ver tabla 2.1), “Crear usuario” (ver tabla 2.6), “Cargar Contenido” (ver tabla 2.4), “Eliminar Usuario” (ver tabla 2.7), “Mostrar Usuario” (ver tabla 2.8), “Enviar Correo” (ver tabla 2.9) y “Efectuar Transacción” (ver tabla 2.10) con lo cual se relaciona esta palabra con la palabra clave *seguridad* y se propone como incumbencia transversal o concern.

También se encuentra repetida la palabra “almacena” en los casos de uso “Crear usuario” (ver tabla 2.6), “Cargar contenido**”** (ver tabla 2.4), “Descargar Contenido” (ver tabla 2.5) y “Eliminar Usuario” (ver tabla 2.7); dicha palabra esta relacionada con la palabra clave *persistencia* y por lo tanto se la propone como concern.

En todos los casos de uso se encuentran presentes las palabras “despliega” y “mostrar” que se pueden relacionar con *visualización* por lo tanto también se propone como concern o incumbencia.

En la siguiente tabla se registra cada incumbencia junto con los casos de uso con los que esta relacionada.

Concern candidato	Casos de Uso	Actor
Verificar	1, 4, 5, 6, 7, 9, 10	Usuario, Administrador
Desplegar	2, 4, 10	Usuario, Administrador
Mostrar	1, 3, 4, 6, 10	Usuario, Administrador
Conectar	1, 4, 5, 6, 7,10	Usuario, Administrador
Almacenar	4, 6, 10	Usuario, Administrador
Seguridad	1, 4, 5, 6, 7, 9, 10	Usuario, Administrador

Tabla Nº 2.11 Primera etapa para la selección de concerns

2.2.5. Tarea 2: Elección de aspectos candidatos

Sub-Tarea 2.1: Identificar crosscutting concerns y concerns funcionales

Para obtener los crosscutting concerns se parte de los concerns de la sub-tarea 1.1 se verifican cuales son requeridos por más de un caso de uso; los que cumplen con esto son: verificar, desplegar, mostrar, conectar, almacenar. Si se observan los crosscutting concerns obtenidos se puede ver que algunos tienen que ser renombrados; tal es el caso de verificar que será cambiado con seguridad, almacenar puede ser renombrado con persistencia, mostrar puede ser asociado con un registro de operaciones o logging.

Sub-Tarea 2.2: Seleccionar aspectos candidatos

En esta sub-tarea se utiliza el dominio del conocimiento y se terminan de seleccionar los *concerns candidatos* que a partir de ahora pasan a llamarse *Aspectos candidatos.* Los aspectos elegidos son los siguientes: Seguridad, Logging o trazas, Almacenamiento temporal (Caching), Persistencia y Tiempo de respuesta (Profiling).

2.2.6. Tarea 3: especificar aspectos candidatos

Sub-Tarea 3.1: Describir responsabilidades

En esta tarea se describe el objetivo y la funcionalidad del aspecto candidato y junto con las dos tareas anteriores se termina de generar la tabla 2.12 que se muestra a continuación:

Aspecto candidato	Casos de Uso	Actor	Responsabilidad
Seguridad	1, 4, 5, 6, 7, 8, 9, 10	Administrador	Restringir el acceso al sistema y a los datos donde sea requerido

Logging o trazas	1, 3, 4, 6, 7, 9, 10	Usuario, Administrador	Imprimir el registro de mensajes que describen las operaciones en ejecución, con propósitos de auditoria o depuración
Almacenamiento temporal (Caching)	2, 3, 9	Usuario, Administrador	Guardar instancias que pueden ser reutilizadas durante la ejecución de la aplicación.
Persistencia	4, 6, 10	Usuario, Administrador	La persistencia permite al programador almacenar, transferir y recuperar el estado de los objetos en un medio secundario para su posterior reconstrucción y utilización.
Tiempo de respuesta (Profiling)	1, 3, 4, 5, 6, 7, 8, 9, 10	Usuario, Administrador	Periodo de tiempo en el cual el sistema debe responder a un servicio, sirve como ayuda para detectar embotellamientos

Tabla Nº 2.12 Aspectos especificados

Sub-Tarea 3.2: Identificar relaciones entre aspectos candidatos y elementos del modelo.

La tabla 2.13 muestra con cuales casos de uso está relacionado un determinado aspecto para posteriormente detectar conflictos.

Casos de Uso / **Aspectos Candidatos**	Caso de uso Login	Caso de Uso Ver contenidos	Caso de Uso Buscar Contenidos	Caso de uso Cargar Contenidos	Caso de uso Descargar Contenidos	Caso de Uso Crear Usuario	Caso de Uso Eliminar Usuario	Caso de Uso Mostrar Usuario	Caso de Uso Enviar Correo	Caso de Uso Efectuar Transacción
Seguridad	X			X	X	X	X	X	X	X
Logging	X		X	X		X	X		X	X
Navegación										
Almacenamiento temporal		X	X						X	
Persistencia				X		X				X
Tiempo de respuesta (Profiling)	X		X	X	X	X	X	X	X	X

Tabla Nº 2.13 Relaciones entre Casos de Uso y Aspectos

2.2.7. Tarea 4: Identificar Conflictos

En esta tarea se identifican a partir de la tabla 2.13 todas las posibles situaciones conflictivas entre concerns. Si más de un aspecto es aplicado sobre un mismo caso de uso, surgirá una situación conflictiva. Esta situación será indicada de forma que pueda ser tratada posteriormente para evitar dichos conflictos. Esto ocurre si la suma de cruces que un caso de uso posee en su columna es mayor que uno.

De acuerdo a la tabla 2.13 se puede ver que todos los casos de uso a excepción del caso de uso Ver Contenidos presentan conflictos ya que la suma de cruces de sus columnas es mayor que uno, esto significa que a la hora de diseñar los aspectos se debe tener cuidado a la hora de aplicarlos a los casos de uso, es decir, se debe dar prioridad sobre cual aspecto es más importante de aplicar para determinado caso de uso.

2.2.8. Diseño (Modelado de Aspectos mediante AML- Aspect Modeling Languaje)

Tarea 5: Modelar en UML

A continuación se describe cada paquete de aspectos con su respectivo conector, el cual encapsula la tecnología de aspectos para la implementación y las reglas de tejido las cuales describen los puntos (Pointcuts) en los cuales se van a aplicar los aspectos y el código o las acciones que se deben ejecutar en dichos puntos (Advices).

El paquete de aspectos provee una representación gráfica (Diagrama de clases) de la vista estática de una incumbencia transversal específica. El paquete base o de componentes puede contener cualquier modelo de UML válido que describe la lógica de negocio del sistema deseado y el conector es el encargado de realizar la interfaz de conexión entre el paquete de aspectos y el paquete de componentes

encapsulando la tecnología que especifica las reglas de tejido. Los aspectos más importantes identificados son los siguientes:

2.2.9. Aspecto de Logging o Trazas.

El aspecto de logging consiste en dejar un registro de las actividades desarrolladas en la ejecución de la aplicación, también ayuda a mirar la interacción entre diferentes partes de un sistema de forma ordenada para entender el comportamiento de un sistema o con propósitos de auditoria o de depuración [34].

Las trazas de código a menudo son duplicadas en muchos partes de una aplicación, conduciendo muchas veces a tener código redundante a través de múltiples componentes en la aplicación. Además las trazas no proporcionan ninguna funcionalidad de la lógica del negocio [34].

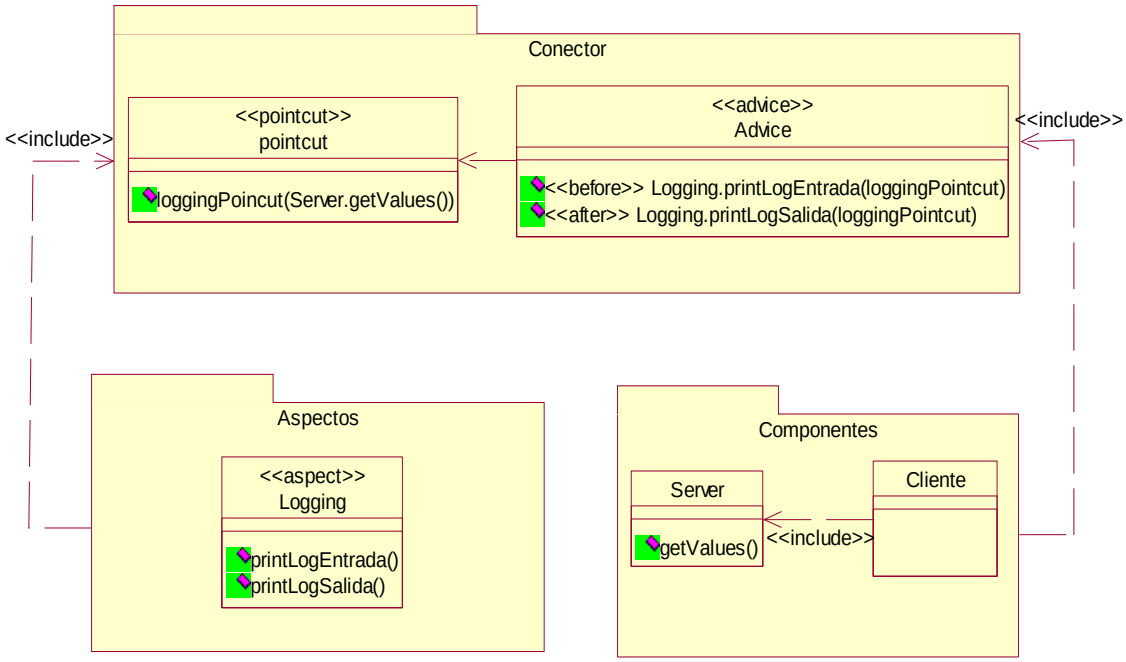

Figura Nº 2.2 Modelado del aspecto de Logging.

En la figura 2.2 la clase Pointcut define los puntos de ejecución en el flujo de control del programa, la clase Advice define el código o las acciones a ser ejecutadas en los pointcuts definidos en la clase Pointcut y la clase Server representa la o las clases del sistema base que van a ser afectadas por el aspecto, todo el modelado de las clases de cada paquete incluido el paquete de aspectos se puede realizar utilizando la aproximación UML tradicional. El paquete conector especifica las reglas de tejido y encapsula la tecnología utilizada, tanto el paquete de aspectos como el de lógica del negocio son independientes entre sí. Cuando el cliente realiza una invocación al Servidor, la acción será registrada (logging), dependiendo de lo que se quiera trazar; puede ser el ingreso de un usuario, una conexión a una base de datos, la confirmación de una transacción bancaria, la ejecución de cualquier método, etc.

El pointcut llamado loggingPointcut se ejecuta siempre que el cliente invoca al método *Server.getValues().* El método *print.LogEntrada()* imprime un registro antes de que se realice la acción y el método print.LogSalida() imprime un registro después de ejecutada la acción que se quiere trazar, no necesariamente se debería aplicar registro a la entrada y salida de un método o conjunto de métodos o constructores.

Si se agregan nuevas clases o métodos al programa sólo se debe cambiar el designador de enlace o pointcut para que dichos métodos generen logging y si se desea generar traza de una manera distinta basta con modificar el consejo o advice. El código relacionado con el logging queda concentrado en un solo lugar, lo cual cumple con la filosofía de modularidad básica, se puede eliminar fácilmente la funcionalidad de traza; eliminando el aspecto.

2.2.10. Aspecto de Seguridad

El aspecto de seguridad se aplica siempre que el usuario realiza una invocación al Servidor que necesita ser autenticada, bien sea el ingreso a una zona restringida o una operación dentro de dicha zona. La figura 2.3 muestra como los pointcuts son activados siempre que el cliente invoca el método *Server.getValues(),* las acciones que son realizadas antes de la llamada al método son leer y comprobar los datos de usuario para autenticarlo en los lugares definidos por los pointcuts [31].

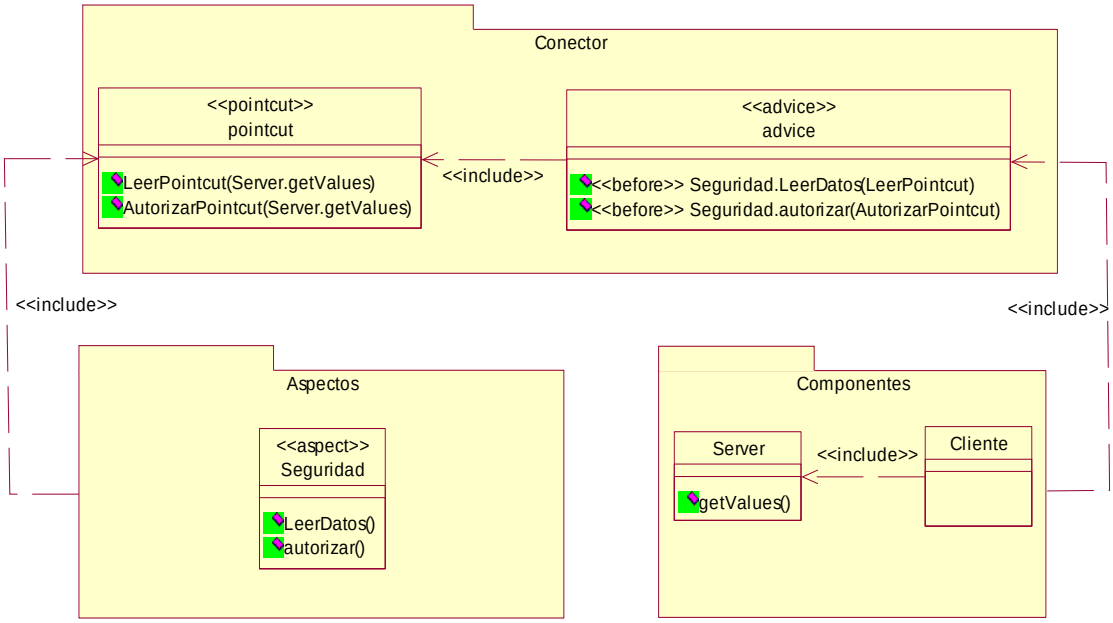

Figura Nº 2.3 Modelado del aspecto de Seguridad.

2.2.11. Aspecto de Almacenamiento Temporal (Caching)

El almacenamiento temporal es un mecanismo para almacenar datos que son utilizados con frecuencia, para reducir al mínimo las llamadas a los métodos encargados de cargar los datos bien sea de una base de datos o de una fuente cualquiera, y resulta en una mejora significativa en el rendimiento de las aplicaciones [47]. También da la posibilidad de actualizar los diferentes tipos de

datos en diferentes intervalos de tiempo (sobre la base predefinida de desalojo y las políticas de actualización de caché).

En las aproximaciones tradicionales los objetos son almacenados en memoria (cache) después de ser usados por primera vez, luego son utilizados en las siguientes llamadas de acceso a datos, sin necesidad de accesar a la fuente nuevamente. Los datos almacenados temporalmente son liberados de la memoria, poniendo en práctica una política de evacuación predefinida, cuando los datos ya no son necesarios.

Para una aproximación basada en Aspectos, primero se escribe un pointcut para interceptar la llamada del método de acceso a datos. Entonces se escribe un advice de tipo "around" para preguntar si los datos del cache corresponden o coinciden con la llamada del método hecha por el cliente. Si no se encuentran dichos datos en el cache los advices se utilizan para llamar al método encargado de accesar a los datos y los datos devueltos se almacenan en la memoria cache y se retornan al cliente. Para las siguientes peticiones se utiliza el mismo pointcut, se revisa si los datos solicitados están en cache a través del advice y se retornan los datos al cliente (ver figura 2.4).

No es fácil introducir o desalojar almacenamiento temporal de forma dinámica cuando hace parte de la lógica del negocio a la cual no pertenece por no cumplir con ninguna funcionalidad de la misma, en cambio a través de la utilización de un aspecto se tiene independencia y se separa completamente del código de la aplicación.

La figura 2.4 muestra una forma de implementar el aspecto de almacenamiento temporal mediante un método *revisarCache(),* el cual es insertado en el punto de corte ubicado en el método *Server.CargarDatos().*

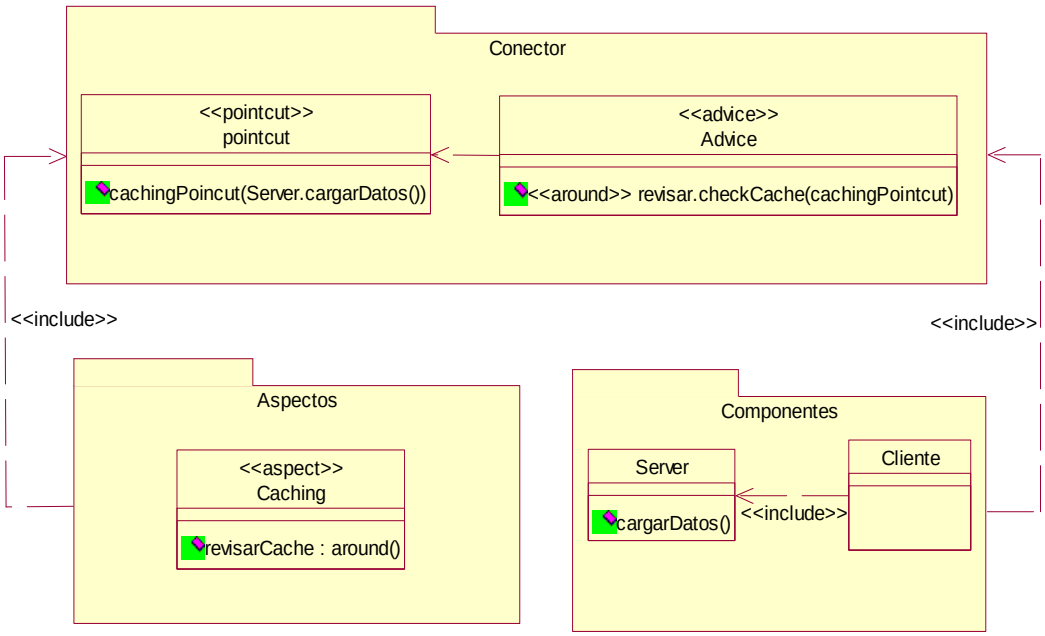

Figura Nº 2.4 Modelado del aspecto de Almacenamiento Temporal.

2.2.12. Aspecto de Profiling.

El aspecto de profiling o tiempo de respuesta permite medir el tiempo que tarda el sistema en responder a determinado servicio [51], por ejemplo el tiempo que tarda en ejecutarse cierta función o cierto grupo de funciones pertenecientes al sistema de componentes. Una de las formas de hacer profiling es obteniendo los tiempos de entrada y salida a una función o método y calculando la diferencia. El aspecto de profiling es una especie de logging que da el beneficio de detectar embotellamientos en la ejecución de una aplicación.

La figura 2.5 muestra una manera de medir el tiempo que tarda en ejecutarse el método *Server.metodo1(),* el aspecto *profiling* contiene los consejos (métodos) para ser aplicados en el punto de corte.

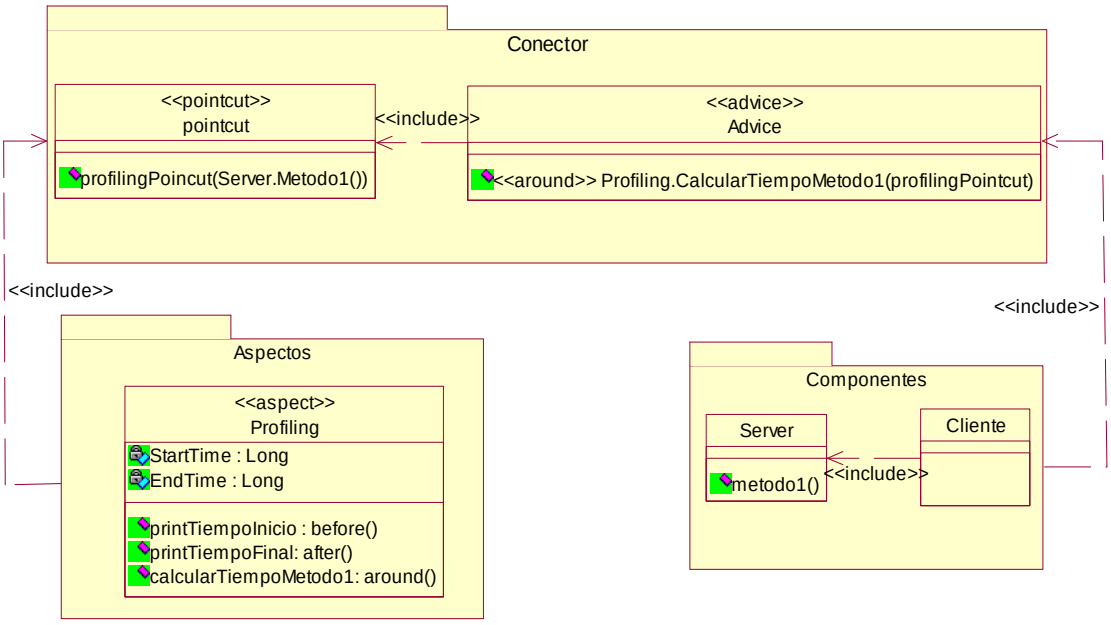

Figura Nº 2.5 Modelado del aspecto de Profiling.

2.2.13. Aspecto de Persistencia.

La persistencia especifica la inserción y la actualización de objetos persistentes. La recuperación y la eliminación de datos persistentes son parte de la lógica de la aplicación, y por lo tanto no pueden ser modularizados en aspectos. Sin embargo, la capa de persistencia proporciona una interfaz estrecha que ayuda a manejar esta incumbencia y aunque se dice que la persistencia hace parte de la lógica del negocio en su totalidad, tal como se entiende un objeto de negocio contiene la lógica del negocio que soporta dicha clase. Otro aspecto es como se guarde esta información en una base de datos para hacerlo persistente [52].

Las instancias de implementación de la interfaz son inyectadas por un aspecto y por lo tanto la implementación es transparente al desarrollador. La capa de persistencia proporcionada es reutilizable y separada de la lógica del negocio (ver figura 2.6).

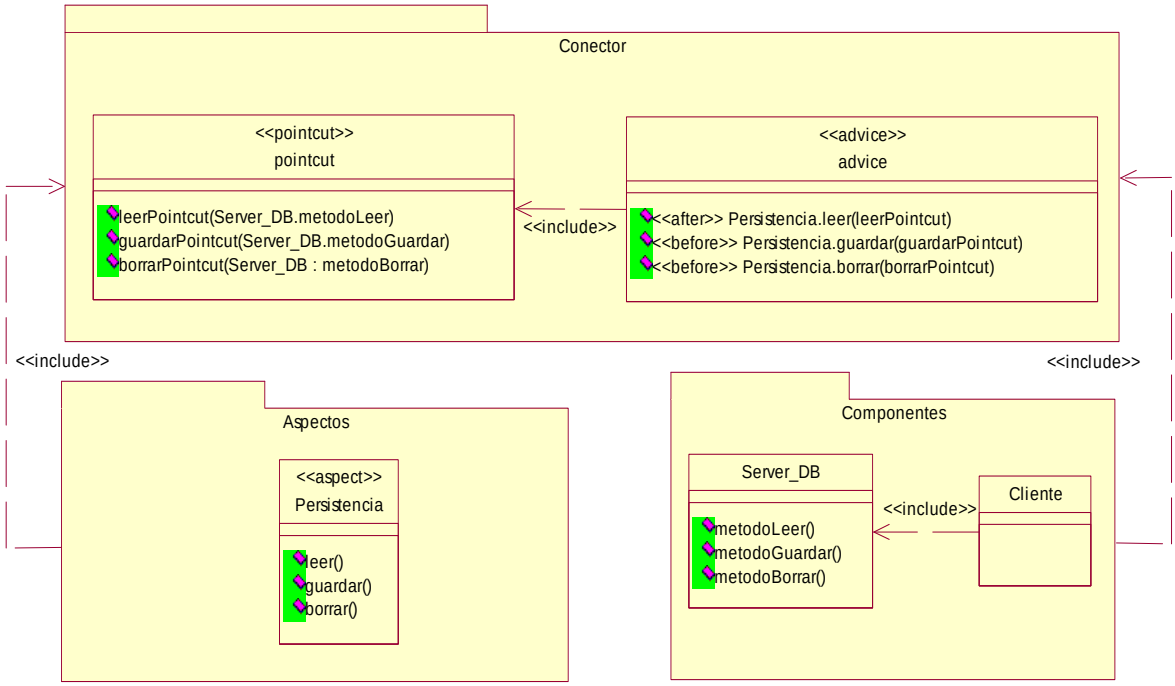

Figura Nº 2.6 Modelado del aspecto de Persistencia.

2.3. Diagrama Arquitectura

La figura 2.7 describe los elementos de la propuesta arquitectónica, la cual puede ser adaptada a cualquier modelo multicapa, por ejemplo el modelo MVC[23] ya que la capa de aspectos es totalmente independiente de cualquier capa de una arquitectura Web; los aspectos encapsulan las incumbencias transversales que se identificaron anteriormente y se agrupan en una única capa de tal manera que ésta capa de aspectos pueda ser asociada con los componentes de la lógica del negocio de cualquier aplicación Web que sea construida con una herramienta orientada a objetos; los aspectos son independientes y no se pueden asociar entre si para evitar conflictos y así cumplir con la filosofía de la Programación Orientada a Aspectos.

[23] Modelo Vista Control

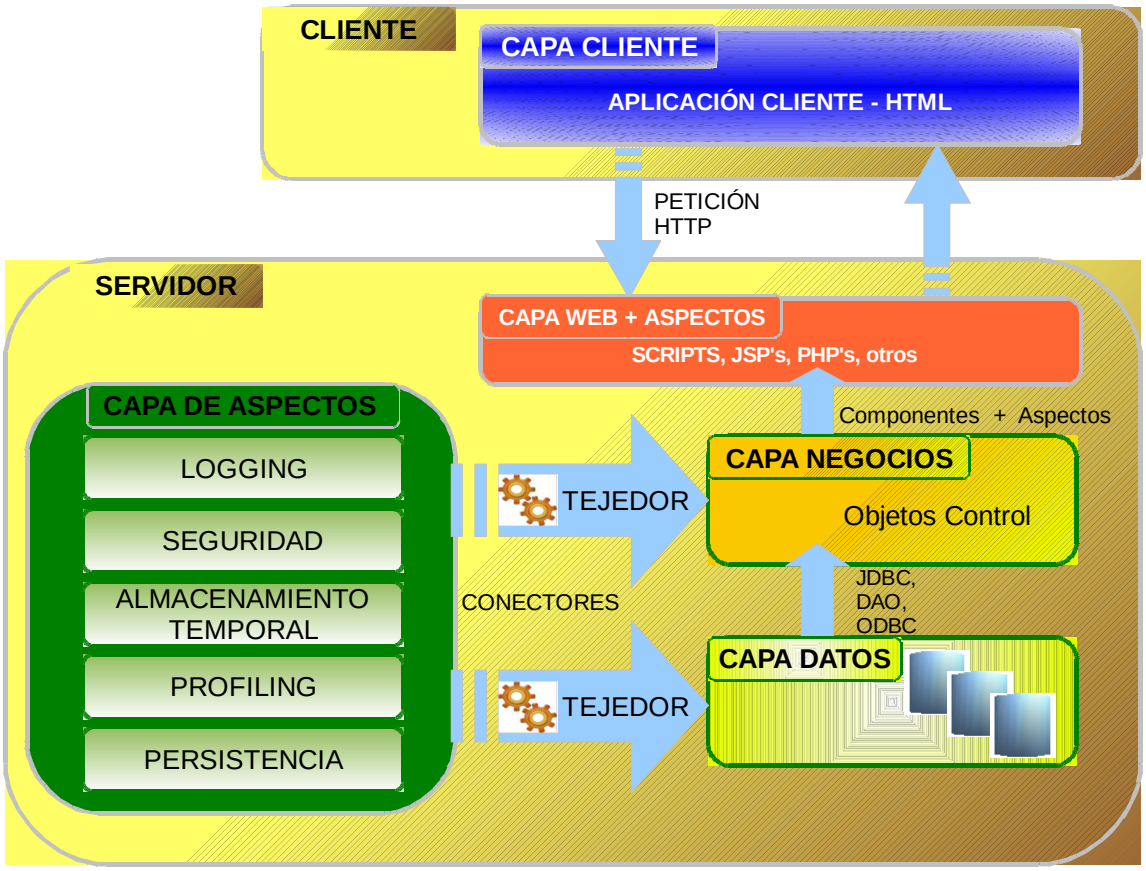

Figura Nº 2.7 Diagrama Arquitectura.

Capa Cliente: Es la capa que presenta el sistema al usuario, le entrega y captura la información en un mínimo tiempo de proceso, se comunica con la capa Web a través de una petición HTTP que se realiza por medio del navegador (browser).

Capa Web: La capa Web esta conformada por scripts como: JSP's, PHP, TCL/TK, etc; es la encargada de dar a los datos un formato adecuado para entregar la información a la capa cliente a través de una respuesta HTML. Envía datos a la capa de negocios a través de formularios que hacen una llamada a la clase *action* (ver figura 1.16), dependiendo de la tecnología solicita y recibe datos de la capa de negocio a través de métodos que invocan los objetos de control; por ejemplo en el caso de tecnología Java los scripts JSP se comunican con servlets de control a través de los métodos *request.getParameter()* o *request.setAttributte().*

Por otro lado esta capa recibe de la capa de negocios los componentes del programa base tejidos con los elementos de la capa de aspectos, de esta forma la capa Web se encarga de dar el formato necesario a los aspectos más los componentes para que puedan ejecutarse en el browser.

Capa de Negocio: En esta capa se procesa la información y se establecen las reglas que se deben cumplir, se comunica con la capa cliente a través de la capa Web y con la capa de datos a través de un protocolo de conexión (JDBC, DAO, ODBC) para solicitar o recuperar datos. No toda la lógica del negocio es la misma, algunas no requieren un frecuente acceso a datos pero una interfaz de usuario robusta necesitará de la lógica de negocio para la validación en la entrada de campos.

La capa de negocio se comunica con la capa de aspectos a través de los conectores los cuales como se mencionó en la sección 2.2.8 encapsulan la tecnología de implementación y establecen las reglas de tejido entre el programa de componentes y el programa de aspectos, definiendo los puntos de enlace y los consejos a aplicarse en dichos puntos.

Capa de Aspectos: La capa de aspectos contiene los aspectos identificados especialmente para una aplicación Web y sirve de base para que se puedan agregar muchos más, por ejemplo los aspectos de sincronización, navegabilidad, manejo de excepciones, etc.

Para agregar un elemento a la capa de aspectos simplemente se debe identificar y encapsular la incumbencia transversal que afecta el resto de capas y se debe especificar el punto de la lógica del negocio donde pueda aplicarse el aspecto (incumbencia modularizada). Independiente de la tecnología de implementación para definir las reglas de tejido entre aspectos y componentes simplemente se deben redefinir los conectores.

Los conectores son los encargados de proporcionar la interfaz entre la capa de aspectos y las capas del sistema base, de este modo, no solo los componentes funcionales serán utilizados como componentes de diseño en la arquitectura, sino también las incumbencias transversales arquitectónicas serán manejadas explícitamente.

Para cambiar de tecnología de desarrollo basta con modificar las reglas de especificación de tejido definidas en los conectores, para el caso especifico de Jboss AOP [32] o de SpringFramework [25] las reglas de tejido se establecen mediante archivos de configuración XML; para el caso de PHP orientado a aspectos [20] las reglas de tejido se definen en el código PHP ya que extiende su sintaxis para implementar aspectos.

Capa de Datos: En esta capa es donde residen los datos y es la encargada de acceder a los mismos, esta formada por uno o más gestores de bases de datos, recibe solicitudes de almacenamiento o de recuperación de datos desde la capa de negocio por medio de un protocolo de acceso a datos.

III. PROTOTIPO DE VALIDACIÓN: CASO PORTAL PARQUE INFORMÁTICO

3.1 Organización Objetivo

La organización objetivo para el prototipo es el Parque Informático Carlos Albán de Popayán utilizando la arquitectura orientada a aspectos propuesta en el capítulo anterior.

3.2 Modelo de desarrollo específico

Para el desarrollo del proyecto se usará el modelo RUP. El Proceso Unificado de Rational (RUP, Rational Unified Process), de manera similar a UML [27], es fruto de los aportes de un gran número de investigadores y empresas de desarrollo de programas. Es un proceso iterativo, centrado en la arquitectura, conducido por casos de uso y soporta técnicas de orientación a objetos en particular el uso de UML. La figura 3.1 muestra la organización del Proceso Unificado de Rational.

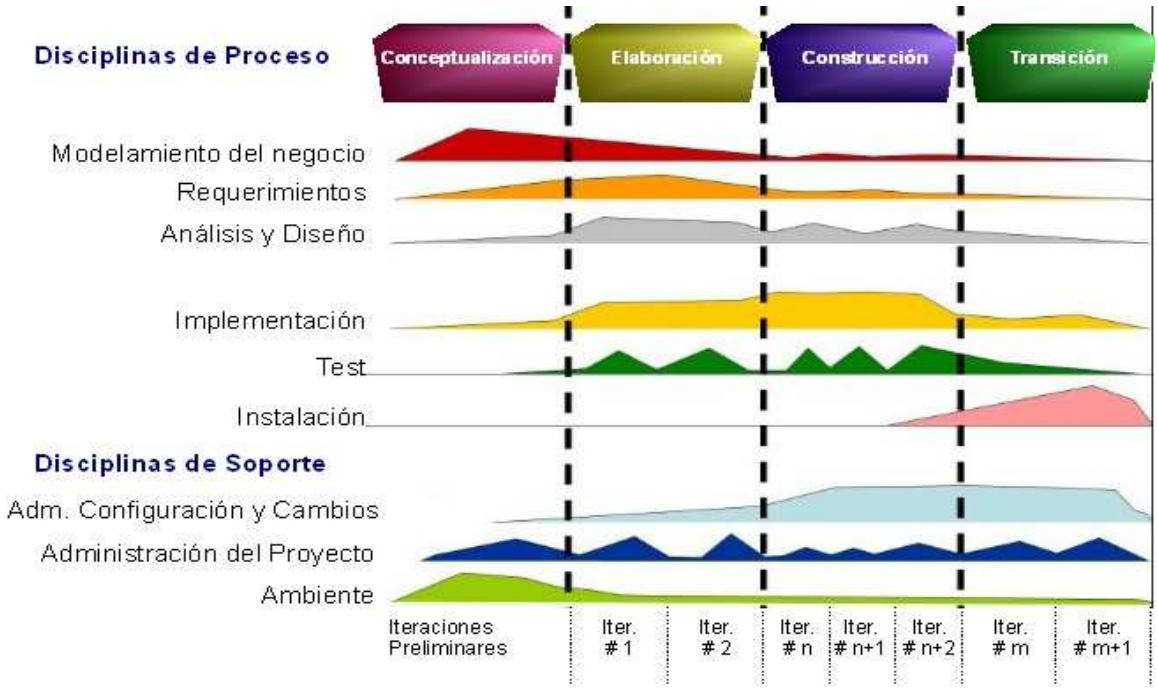

Figura Nº 3.1 Organización del Proceso Unificado

3.3. Modelo Del Negocio

3.3.1 Descripción Parque Informático de Ciencia y Tecnología

El Parque Informático de Ciencia, Arte y Tecnología "Carlos Albán"[24] es una institución que brinda un espacio para las Ciencias, la tecnología informática y los medios de comunicación. Esta compuesto por ocho dependencias; las cuales ofrecen diferentes cursos, realizan diferentes eventos y actividades dirigidas hacia la comunidad de los estratos bajos de la ciudad de Popayán.

Actualmente el Parque Informático de Popayán da a conocer la información sobre las dependencias que lo conforman, su misión, visión, calendario de inscripciones a cursos, eventos, actividades y calificaciones de los estudiantes a través de carteleras y de plegables repartidos en la misma institución por la secretaría del Parque, dicha oficina se encarga de recibir y publicar la información de cada encargado de su respectiva sala y las calificaciones de los profesores de cada curso.

3.3.2. Actores del Negocio

Diagrama Casos de Uso Responsable Sala

El negocio se inicia cuando la persona encargada de administrar alguna de las salas del Parque Informático desea publicar información de su dependencia (ver figura 3.2).

[24] Información obtenida a través de la Jefe de Sistemas del Parque Informático Carlos Albán de Popayán (Calle 5 entre Carreras 23 y 24 Telf. 8316169)

Figura Nº 3.2 Diagrama de Casos de Uso Responsable Sala (Modelo de la Organización)

Diagrama Casos de Uso Profesor

El profesor de cada curso entrega las calificaciones de sus estudiantes a la Secretaría del Parque Informático para que sean publicadas (ver figura 3.3).

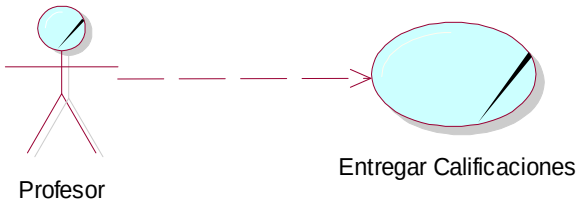

Figura Nº 3.3 Diagrama de Casos de Uso Profesor (Modelo de la Organización)

Diagrama de Caso de Uso de la Secretaría

La secretaría del Parque Informático recibe información de cada dependencia; puede ser calendario de eventos, actividades y/o calificaciones de los profesores de cada curso (ver figura 3.4).

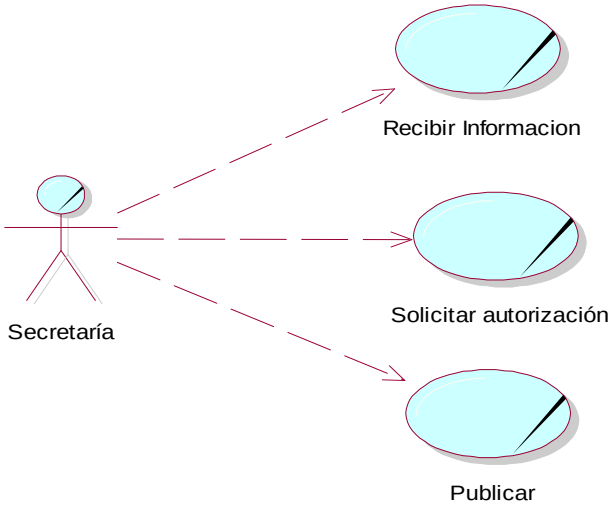

Figura Nº 3.4 Diagrama de Casos de Uso de Secretaría (Modelo de la Organización)

Diagrama Casos de Uso del Director del Parque Informático

El director del Parque Informático es el encargado de revisar la información que reciba de la secretaria y de autorizar su publicación (ver figura 3.5)

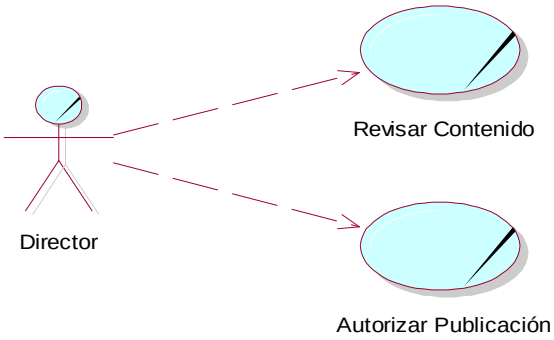

Figura Nº 3.5 Diagrama de Casos de Uso del Director (Modelo de la Organización)

3.4. Captura De Requisitos

3.4.1 Descripción del Sistema a Implementar

El prototipo a desarrollar es una aplicación Web basada en la arquitectura orientada a aspectos definida en el capítulo anterior, lo primero que se desarrolla es la parte de la lógica del negocio o sistema de componentes, la cual será implementada utilizando tecnología Java (JSP's/Servlets), luego se le agregarán tres de los aspectos definidos en la arquitectura propuesta anteriormente para poder integrarla en un solo sistema, se utiliza el framework Jboss AOP integrado con el Servidor Jboss AS y servidor de bases de datos MySQL.

El portal permitirá a los encargados de cada dependencia (ver tabla 3.1) actualizar la información que deseen publicar por medio de un gestor de contenidos; dicha información será almacenada en una base de datos y será consultada por los usuarios del Portal. Para acceder al gestor de contenidos se solicitará una identificación (Login y Password) el sistema realiza la conexión para verificar que los datos existan en el servidor de bases de datos, una vez realizada la validación se ofrecerá un menú para que el administrador pueda seleccionar el tipo de información que va a actualizar. Posteriormente serán insertados los aspectos de Logging, Seguridad y Profiling implementados mediante el framework Jboss AOP.

La razón por la cual el prototipo no se implementó utilizando phpAspect tal y como se había planteado al inicio del proyecto es porque se concluyó que la herramienta es bastante inestable; una vez instalada mediante las dependencias que requería: librerias de PEAR (Php Extension and Repository); php Parse Tree, Php beautifier y Console Getopt se puede correr desde la línea de comandos y efectuar el tejido entre el código Php de componentes y el código de aspectos de phpAspect; sin embargo al realizar el tejido la herramienta es bastante inestable y genera muchos errores por ser una versión beta, actualmente su autor William Candillon está desarrollando un plugin para Eclipse el cual seguramente podrá ofrecer toda la potencialidad de AOP a través de una sintaxis extendida del lenguaje Php y

basada en el lenguaje más poderoso actualmente para aspectos que es AspectJ. Estudiando otras alternativas para implementar aspectos en aplicaciones Web la más adecuada es Jboss AOP.

3.4.2. Actores

La tabla 3.1 describe los actores del sistema.

ACTOR	DESCRIPCION
Administrador	Se encarga de actualizar todos los datos del sistema como adicionar usuarios, administrar usuarios y sus permisos.
Responsable Sala	Es el encargado de actualizar los contenidos de su correspondiente sala del Parque Informático.
Profesor	Es el encargado de subir las calificaciones de los estudiantes al portal Web.
Estudiante	Es la persona que hace uso de los servicios del portal; tales como consulta de calendarios, actividades, eventos, inscripciones a cursos o consulta de calificaciones.

Tabla Nº 3.1 Descripción de actores

3.4.3. Descripción casos de uso de alto nivel

La tabla 3.2 describe los casos de uso iniciales:

Caso de Uso	Gestionar Sesión
Actor	Administrador, Responsable Sala, Profesor
Descripción	Permite al Administrador y/o Responsable Sala poder iniciar y finalizar una sesión, previa autenticación.

Caso de Uso	Administrar Usuario
Actor	Administrador
Descripción	Permite actualizar la información de las cuentas de los responsables de las salas y de los profesores, adicionar, editar o eliminar.

Caso de Uso	Administrar contenidos
Actor	Responsable Sala
Descripción	Permite agregar, editar o eliminar contenido de una dependencia.

Caso de Uso	Administrar Calificaciones
Actor	Profesor
Descripción	Permite a un profesor de determinado curso poder publicar, editar o eliminar las calificaciones de los estudiantes en la aplicación

Caso de Uso	Gestionar Estudiante
Actor	Estudiante
Descripción	Permite a un usuario de la aplicación poder inscribirse en uno de los cursos del Parque Informático y/o consultar sus calificaciones a través del portal.

Tabla Nº 3.2 Descripción Casos de Uso de Alto nivel.

Diagrama Inicial Casos de Uso del Administrador

El Administrador estará a cargo de la gestión de los usuarios una vez haya iniciado sesión (ver figura 3.6).

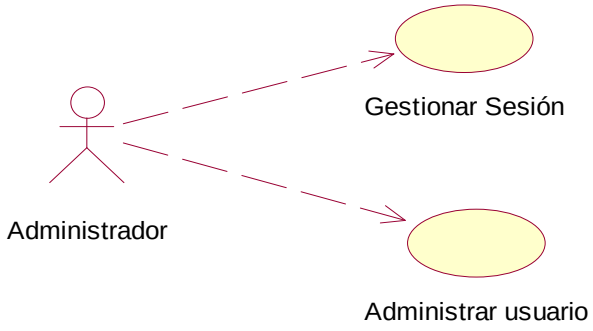

Figura Nº 3.6 Diagrama de Casos de Uso del Administrador

Diagrama Casos de Uso del Responsable Sala

La persona encargada de cada dependencia estará a cargo de administrar el contenido respectivo (ver figura 3.7)

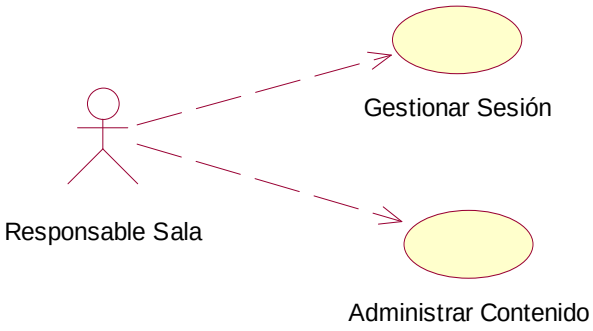

Figura Nº 3.7 Diagrama de Casos de Uso del Responsable Sala

Diagrama Casos de Uso del Profesor

Cada profesor se encargará de gestionar las calificaciones de sus estudiantes (ver figura 3.8)

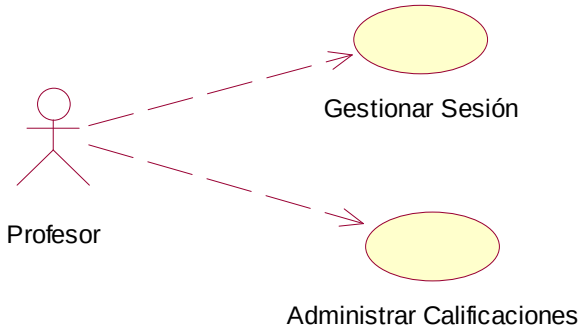

Figura Nº 3.8 Diagrama de Casos de Uso del Profesor

Diagrama Casos de Uso del Estudiante

El estudiante podrá gestionar información de su interés (ver figura 3.9)

Figura Nº 3.9 Diagrama de Casos de Uso del Estudiante

3.5. Análisis De Riesgos

En la tabla 3.3 se presentan los posibles riesgos para el desarrollo del prototipo y las soluciones para evitarlos y/o solucionarlos.

Riesgo:	Requisitos incompletos.
Descripción:	El sistema no satisface todos los requisitos planteados al inicio.
Impacto:	Incumplimiento del objetivo general y/o bajo nivel de calidad del producto.
Estrategia de Eliminación:	Elaborar un buen modelado y diseño de la aplicación.

Estrategia de Minimización:	Evitar realizar cambios en mitad del desarrollo de la aplicación.
Plan de Contingencia:	Solicitar todos y cada uno de los requisitos al inicio del desarrollo.

Riesgo:	Dificultad de los usuario para utilizar el producto
Descripción:	Todos los usuarios no pueden utilizar el servicio porque no están familiarizados con las aplicaciones Web.
Impacto:	Los usuarios prefieren el antiguo sistema.
Estrategia de Eliminación:	Informar e instruir a los usuarios.
Estrategia de Minimización:	Realizar un seguimiento de como los usuarios están utilizando la aplicación Web
Plan de Contingencia:	Al implantar la aplicación entregar al administrador y a los usuarios un folleto o manual donde se informe detalladamente como usar la aplicación.

Tabla Nº 3.3 Descripción de Riesgos

3.6. Casos de Uso Extendidos

3.6.1 Modelo de Casos de Uso:

La siguiente figura muestra los casos de uso extendidos de los casos de uso de alto nivel.

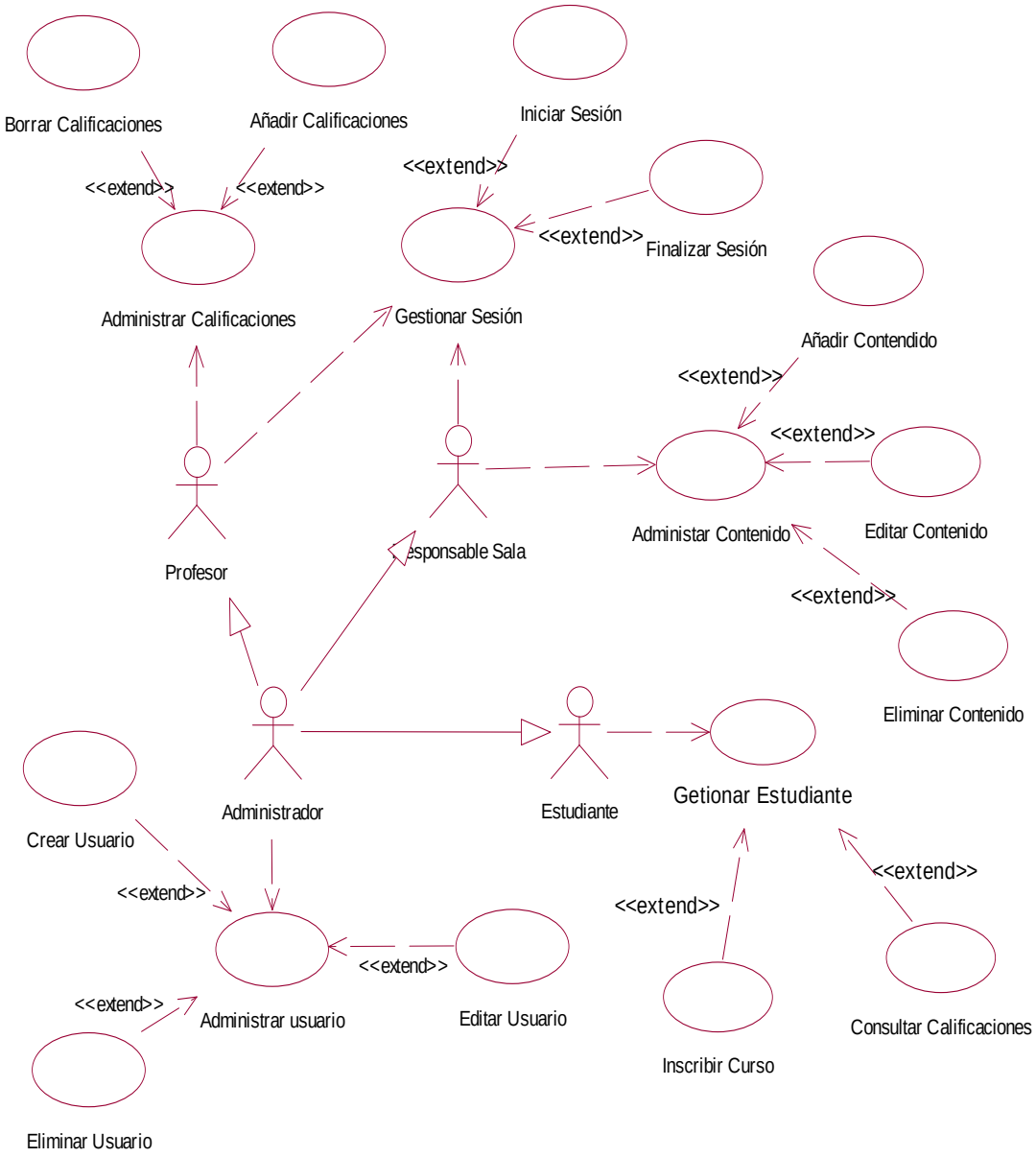

Figura Nº 3.10 Diagrama Casos de Uso Extendidos

3.6.2. Descripción Casos de Uso Extendidos

A continuación se realiza la especificación de los casos de uso extendidos más relevantes (ver tablas: 3.4, 3.5, 3.6 y 3.7)

Caso de Uso	**Iniciar Sesión**	
Descripción	Permite a un usuario registrado ingresar a cierto sitio de la aplicación Web, previa verificación de login y password.	
Actor	Administrador, Responsable Sala, Profesor	
Trigger	El usuario selecciona la opción ingresar	
Flujo de Eventos	**Flujo Básico**	El sistema verifica que el login y password sean correctos
		El sistema permite el acceso al usuario mostrando el sitio restringido
	Flujo Alternativo	Si los datos de validación ingresados por el usuario son incorrectos el sistema muestra un mensaje de error y se reinicia el caso de Uso
Requisitos especiales	Conexión segura a la base de datos de usuarios.	
Suposición	Debe existir un registro de login y password en la base de datos	
Post_Condiciones		
Puntos de extensión		

Tabla Nº 3.4 Descripción Caso de Uso Iniciar Sesión

Caso de Uso	**Crear Usuario**	
Descripción	Permite a un administrador crear usuarios y asignar permisos para ingreso a la aplicación	
Actor	Administrador	
Trigger	El administrador selecciona la opción Crear usuarios.	
Flujo de Eventos	**Flujo Básico**	El sistema verifica que el administrador este autorizado para realizar tal operación
		El sistema despliega un formulario de registro y un menú de opciones para fijar las características del usuario.

		El sistema almacena la información del usuario.
	Flujo Alternativo	Si el usuario a crear ya existe el sistema muestra un mensaje de error y se reinicia el caso de Uso.
Requisitos especiales	Conexión segura a la base de datos de usuarios.	
Suposición	El administrador debe haber ingresado	
Post_Condiciones		
Puntos de extensión	El sistema confirma los resultados obtenidos mediante el caso de uso mostrar usuarios	

Tabla Nº 3.5 Descripción Caso de Uso Crear Usuarios

Caso de Uso	**Añadir Contenido**	
Descripción	Permite a las personas responsables de cada sala agregar contenidos a la aplicación Web.	
Actor	Responsable Sala	
Trigger	El Encargado de cada sala selecciona añadir contenido	
Flujo de Eventos	**Flujo Básico**	El sistema verifica que el Responsable de la Sala este autorizado para realizar tal operación
		El sistema despliega un formulario y un menú de opciones para fijar los contenidos.
		El sistema almacena la información de contenidos.
	Flujo Alternativo	Si los datos ingresados no son válidos el sistema despliega un mensaje de error y se reinicia el caso de Uso.
Requisitos especiales	Conexión segura a la base de datos de contenidos.	
Suposición	Responsable de Sala debe estar autenticado	
Post_Condiciones		
Puntos de extensión		

Tabla Nº 3.6 Descripción Caso de Uso Añadir Contenidos

Caso de Uso	**Añadir Calificaciones**	
Descripción	Permite a un Profesor subir las calificaciones al portal Web.	
Actor	Profesor	
Trigger	El Profesor selecciona la opción Añadir Calificaciones	
Flujo de Eventos	**Flujo Básico**	El sistema despliega un menú de opciones para escoger el curso.
		El sistema almacena la información de calificaciones.
	Flujo Alternativo	Si el profesor escoge un curso que no le corresponde el sistema muestra mensaje de error.
Requisitos especiales	Conexión segura a la base de datos de Calificaciones.	
Suposición	El profesor debe estar autenticado	
Post_Condiciones		
Puntos de extensión		

Tabla Nº 3.7 Descripción Caso de Uso Añadir Calificaciones

3.7. Análisis

3.7.1. Construcción de la Vista Lógica

- **Diagrama general de clases**

A partir de los casos de uso de alto nivel (ver tabla 3.2) se obtienen un diagrama de clases generales (ver figura 3.11), las cuales se relacionan mediante asociaciones.

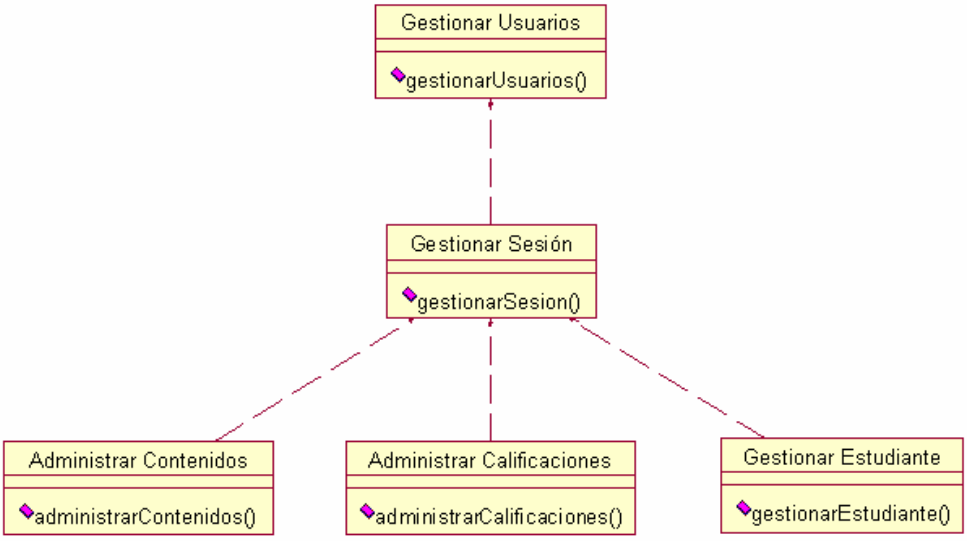

Figura Nº 3.11 Diagrama General de Clases

3.8 Diseño

3.8.1 Diagrama Clases Diseño

A partir de los casos de uso extendido (ver figura 3.10) se especifican las clases de diseño mostradas en la figura 3.12.

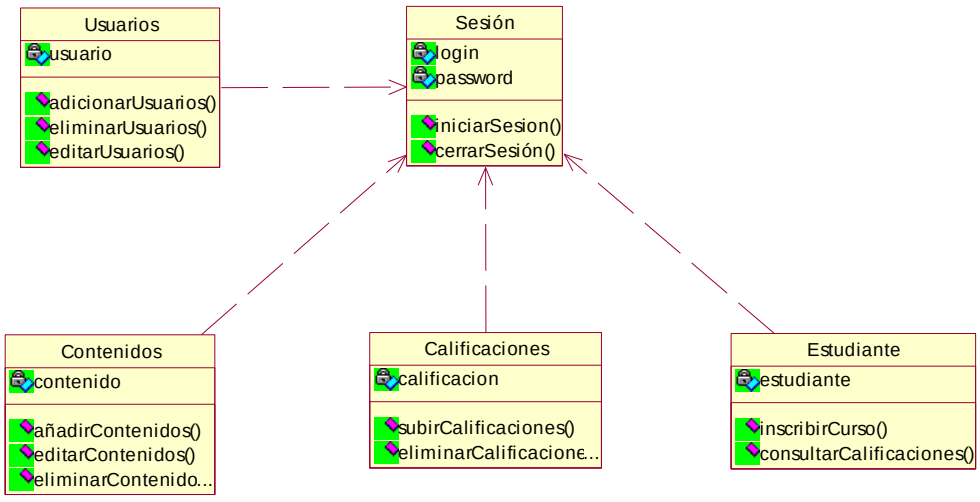

Figura Nº 3.12 Diagrama Clases Diseño

3.8.2 Diagrama de Paquetes

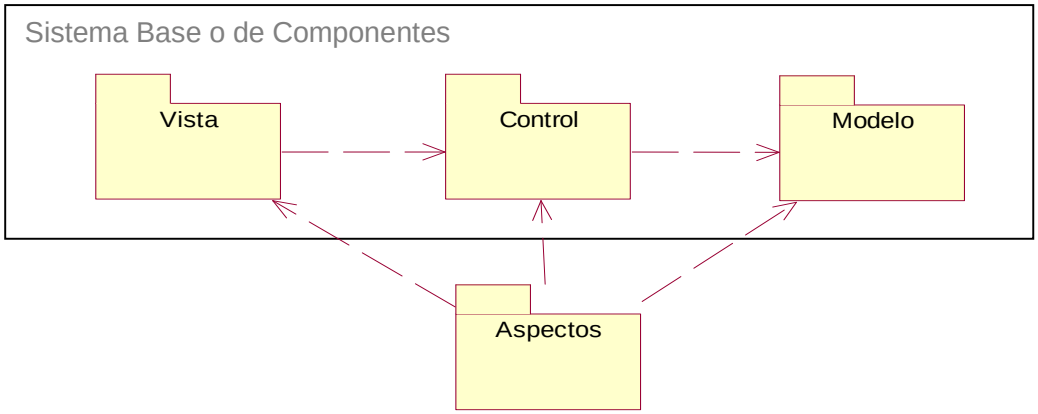

Figura Nº 3.13 Diagrama de Paquetes

El paquete Vista agrupa todas las clases que manejan las interfaces gráficas de usuario, en este caso a través de archivos JSP's [28] [29], el paquete Control; formado por las clases que reciben la petición del lado del cliente en clases tipo Servlet [30] y se encargan de invocar el comando específico asociado a la operación, el paquete Modelo, donde están la clases o Beans encargados de realizar los servicios de acceso a datos y el paquete de aspectos donde se encapsulan las incumbencias transversales que se comunican con el resto del sistema a través de las reglas de especificación de tejido (Pointcuts y Advices) (ver figura 3.13).

3.9 Implementación

3.9.1 Diagrama Clases de Implementación

La figura 3.14 muestra las diferentes clases para la implementación del portal aplicando el patrón Modelo - Vista - Controlador. Los Servlets se encargan de la lógica y del control de navegación; los Beans son clases con funciones más

específicas, como el acceso a datos y los Jsp's, son páginas de presentación que interactúan con el usuario.

Inicialmente se tiene la clase index.html, que se encarga de pedir los datos login y password, por medio de un formulario, estos son enviados al servlet Usuario para su autenticación y dependiendo del perfil o tipo de usuario (administrador, Responsable Sala, Profesor o Estudiante) se carga una página de administración (administrador.jsp), la cual se conecta a un servletAdministrador, éste a su vez permite gestionar el contenido de la aplicación Web dependiendo del perfil de ingreso.

En este caso solo se representa el caso del Administrador, ya que el modelo es igual para los otros perfiles como gestión de contenido; en caso de que el perfil con el que se ingresó sea Responsable de Sala, gestión de calificaciones; en caso de ser un Profesor o gestión cursos y consulta calificaciones en caso de ser un estudiante.

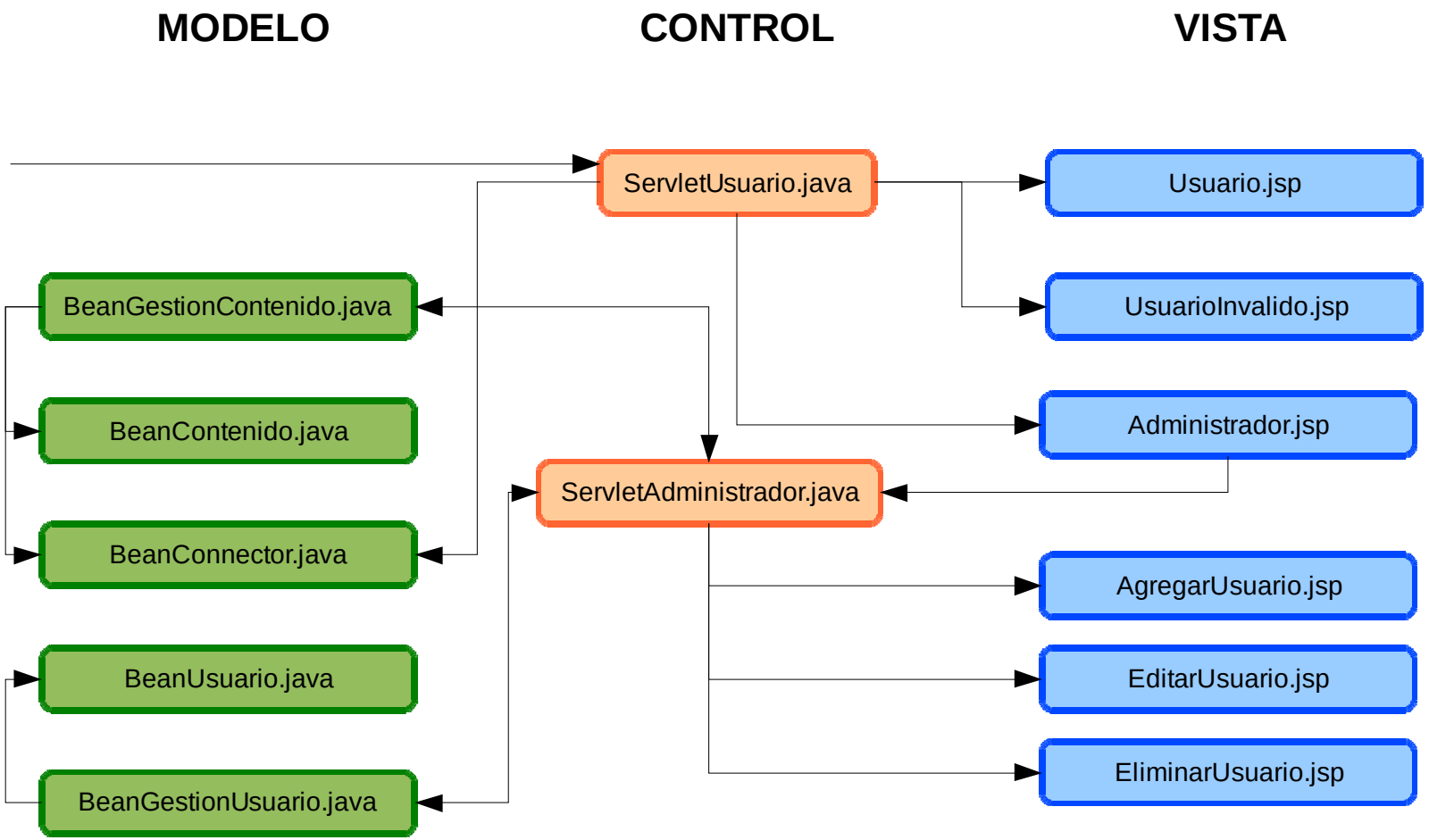

Figura Nº 3.14 Diagrama Clases Implementación

3.10. Aplicación de la Arquitectura de Aspectos

Para el caso de estudio se van a implementar los aspectos de logging, seguridad y profiling dentro de un paquete llamado aspectos como capa adicional definida en la arquitectura orientada a aspectos y se va a hacer relación de cómo los elementos generales definidos en la arquitectura se pueden particularizar en un caso específico, para probar que la arquitectura diseñada puede ser aplicada.

En este caso se utiliza la herramienta JBoss AOP la cual implementa los advices usando interceptores (*interceptors*) pertenecientes al paquete *org.jboss.aop.** De tal forma que se pueden definir consejos que intercepten métodos invocados, constructores o propiedades de una clase.

JBoss AOP ofrece dos maneras de tejer los Join Points y los advices. Una es usando un archivo XML externo de configuración llamado jboss-aop.xml, este archivo define el tejido entre las clases del código fuente con las clases de los aspectos, es decir, los pointcuts o puntos en donde se desean insertar los advices; clases a ser afectadas y métodos de dichas clases y la clase interceptora implementada en Jboss AOP, dependiendo del Framework utilizado pueden cambiar las palabras de los tags del archivo XML, AspectWerks también utiliza un archivo de configuración XML para el enlazado. Otra forma de enlazar en Jboss AOP es usando las anotaciones (*annotations*) Java [31].

3.10.1. Aplicación del aspecto de Logging.

Para el caso de estudio se desea aplicar Logging para registrar la conexión a la base de datos, es decir la llamada a los métodos de la clases encargadas de realizar la conexión, en este caso la función *conectarBD()* de los servlets ServletAdministrador, ServletContenido y ServletCalificaciones. Como se mencionó en el Capítulo II el paquete Conector encapsula la tecnología con la cual se implementan los aspectos, en este caso Jboss AOP; la clase loggingInterceptor que es el inicio de la creación del aspecto debe implementar la

interfase ***jboss.aop.Interface***; El advice o consejo llamado *System.out.print("Conexión efectuada!!!");* utiliza un ***Interceptor,*** el cual no es otra cosa que un aspecto con un solo advice perteneciente al paquete ***org.jboss.aop*** (ver figura 3.15)***.***

En el archivo de configuración XML se definen los lugares (*pointcuts*) donde se va a enlazar el aspecto con la funcionalidad base, es decir, las operaciones o métodos los cuales requieren el servicio de Logging, en este caso, *conectarBD()* pertenecientes a los Servlets que realizan conexiones.

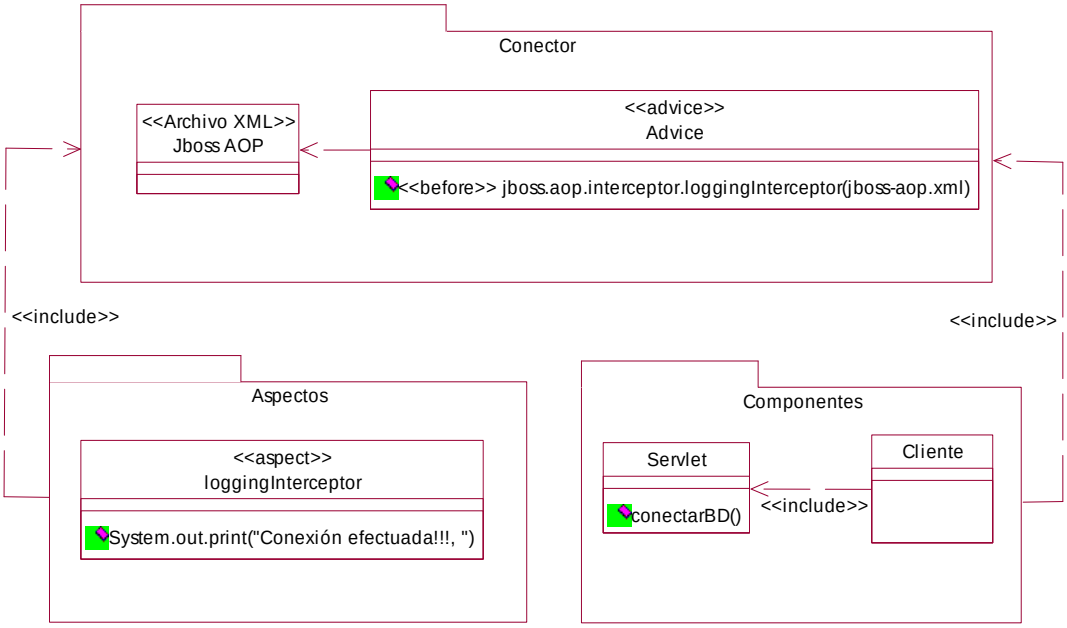

Figura Nº 3.15 Implementación del aspecto de Logging

A continuación se muestra la estructura del archivo jboss-aop.xml para el aspecto de Logging aplicado al método *ServlerAdministrador.ConectarBD()* y *ServlerContenido.ConectarBD().*

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<aop>
    <bind pointcut="execution(private void control.ServletContenido-
&gt;conectarBD())">
        <interceptor class="aspectos.loggingInterceptor"/>
    </bind>
    <bind pointcut="execution(private void control.ServletAdministrador-
&gt;conectarBD())">
        <interceptor class="aspectos.loggingInterceptor"/>
    </bind>
</aop>
```

3.10.2. Aplicación del aspecto de Profiling.

Bajo JBOSS AOP, la clase profilingInterceptor cobija al método BeanGestionUsuario.InsertarUsuario(), cuando se invoca a dicho método, el framework AOP rompe la llamada de método en sus partes y encapsula dichas partes en un objeto de Invocación. El aspecto envuelve y delega al método actual y mediante los atributos del aspecto se definen los tiempos de inicio y fin de ejecución para realizar la medición. Luego se obtiene información contextual sobre la llamada de método del objeto de Invocación, y finalmente se imprime la duración en milisegundos mediante la diferencia entre los tiempos actual y de inicio (ver figura 3.16).

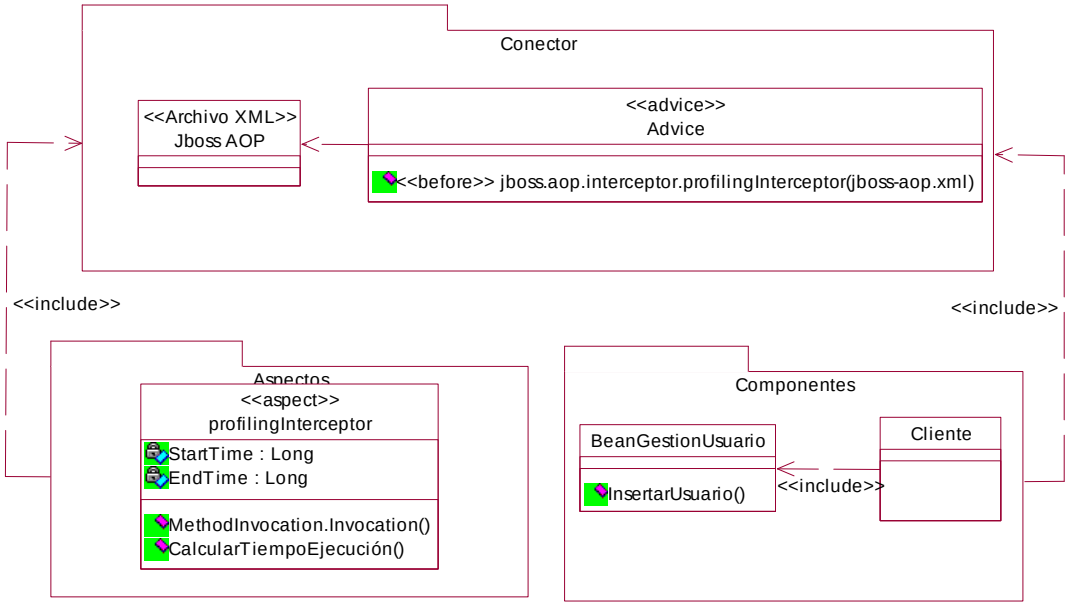

Figura Nº 3.16 Implementación del aspecto de Profiling

A continuación se muestra la estructura del archivo jboss-aop.xml para el aspecto de profiling aplicado al método componentes.BeanGestionUsuario.InsertarUsuario.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<aop>
    <bind pointcut="execution(public boolean
componentes.BeanGestionUsuario-&gt;InsertarUsuario(java.lang.String,
java.lang.String, java.lang.String, java.lang.String, java.lang.String,
java.lang.String))">
        <interceptor class="aspectos.profilingInterceptor"/>
    </bind>
</aop>
```

3.10.3. Aplicación del aspecto de Seguridad.

Se aplica el aspecto de seguridad para el control de acceso a las zonas restringidas de la aplicación Web cada vez que se invoque un servicio de la clase ServletUsuario *(HttpServletRequest request, HttpServletResponse response)*. La clase seguridad encargada de la creación del aspecto debe implementar la interfaz del paquete *jboss.aop* tal como lo hizo la clase Logging en el aspecto anterior; el

archivo de configuración XML es el encargado de definir las operaciones de la aplicación que invocan el control de acceso (lugares en la ejecución del programa donde se aplican los advices) (ver figura 3.17).

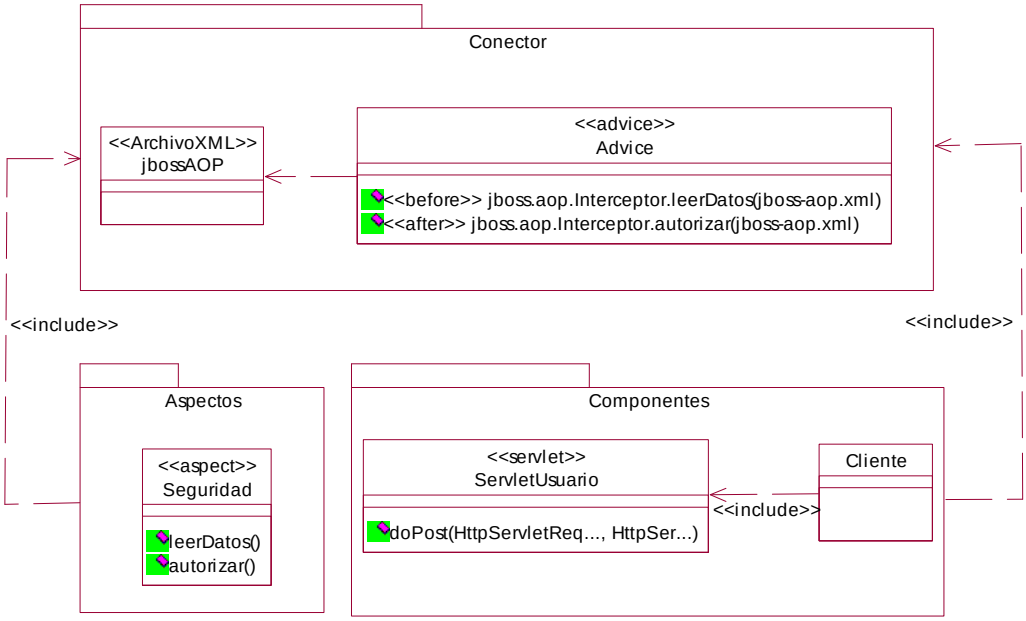

Figura Nº 3.17 Implementación del aspecto de Seguridad

Las reglas de tejido para el aspecto de seguridad se definen en el siguiente código XML.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<aop>
    <bind pointcut="execution(public void vista.ServletUsuario-
&gt;doPost(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse))">
        <interceptor class="aspectos.seguridadInterceptor"/>
    </bind>
</aop>
```

3.11. Pruebas

Las actividades realizadas durante esta fase buscan verificar que la arquitectura pueda ser aplicada al prototipo y corregir los posibles errores o conflictos que pudieran surgir al integrar aspectos con componentes funcionales (Casos de Uso). Las actividades que se consideran importantes en esta fase se presentan a continuación.

Elaboración del plan de pruebas de integración: consiste básicamente en elaborar un plan de pruebas que permita verificar el cumplimiento de cada uno de los casos de uso definidos en la etapa de requisitos y el funcionamiento en conjunto al tejerlos con los elementos de la capa de aspectos, teniendo en cuenta para cada prueba el nombre, la descripción, los resultados esperados y los resultados obtenidos.

Aplicación del plan de pruebas: consiste en realizar las pruebas descritas en el plan al prototipo construido. Cuando los resultados esperados difieran de los resultados obtenidos, debería elaborarse una lista de errores identificando el caso de uso o aspecto causante del conflicto, los parámetros de entrada proporcionados a ese elemento, el resultado que se obtuvo y el resultado que se esperaba.

Corrección de errores: consiste en corregir cada uno de los errores que aparezca en la lista de errores producto de la aplicación del plan de pruebas.

Actualización de modelos: debido a que la corrección de errores puede traer consigo una nueva definición de objetos, clases, métodos y en general modificación del código, no solo en la parte de componentes sino en la parte aspectual (modificación de puntos de enlace y consejos) es necesario actualizar los modelos para mantener la consistencia entre estos y la implementación finalmente realizada.

IV. CONCLUSIONES

- La programación orientada a objetos durante años ha sido una de las herramientas más adecuadas para diseñar software, sin embargo todavía existen algunas funcionalidades que se escapan a su encapsulamiento, es entonces donde entra a complementarla la programación orientada a aspectos. Si bien es cierto que no es una herramienta muy difundida hoy en día muchos de los lenguajes de programación la han adoptado creando extensiones, frameworks y herramientas para poder soportarla, ejemplo AspectJ y Jboss AOP en el caso de Java.

- A pesar de que adoptar una nueva metodología de diseño implica muchos recursos tanto humanos como materiales, hoy en día DSOA tiene una comunidad muy amplia que aporta investigación; para unificar la Ingeniería del software de aspectos en una sola aproximación oficial utilizando el estándar UML, de tal forma que la adopción de AOP sea bastante sencilla para los que tienen conocimiento de desarrollo de software orientado a objetos.

- La Arquitectura Orientada a Aspectos se diseño mediante la aproximación AML, la cual encapsula en paquetes los aspectos y componentes por separado y utiliza el concepto de conectores introducido en la versión 2.0 de UML, si bien es cierto que los lenguajes de descripción arquitectónicos manejan un concepto diferente de arquitectura, en última instancia están compuestos por dos elementos: composición e interacción; por lo tanto se puede pensar en UML como una alternativa para diseñar software desde un nivel arquitectónico.

- A pesar de que AspectJ es el lenguaje más potente para implementar aspectos por ampliar la sintaxis de Java aún no puede ser utilizado para desarrollar

aplicaciones Web; la razón es que su compilador llamado AJC no es compatible con el compilador de los JSP's / Servlets (JDK).

- Aunque se pueden implementar muchos aspectos en una aplicación Web, la arquitectura propuesta se diseño como base y punto de partida para agregarle más aspectos, como por ejemplo el aspecto de navegación muy estudiado hoy en día por investigadores, de forma tal que dichos aspectos sean reutilizables y completamente separados del sistema base y separados entre sí para cumplir con la filosofía de AOP.

- Los aspectos pueden extenderse desde conceptos de alto nivel como seguridad y calidad del servicio hasta conceptos de bajo nivel como autenticación, almacenamiento temporal, buffering, etc. Pueden ser funcionales, como características o reglas de negocio, o no funcionales, como sincronización y manejo de transacciones.

- AOP es fácil de adaptar ya que cualquier diseñador de software que esté familiarizado con tecnologías orientadas a objetos puede apropiarse de una herramienta UML que soporte aspectos y así diseñar aplicaciones mediante el estándar UML a la par con el modelado de aspectos.

- Aunque AOP ha recibido fuertes críticas diciendo que se exagera su importancia y que no permite hacer nada que no se pueda implementar con la orientación a objetos, la ventaja principal de AOP no esta en lo que hace sino en como lo hace, consiguiendo un código más fácil de depurar y más fácil de mantener, como se probó en la parte del desarrollo del prototipo:
  El caso del aspecto de logging el cual requería imprimir un registro siempre que hubiera conexión con la base de datos, esta incumbencia fue implementada en un aspecto y los métodos que realizaban esta operación fueron sacados de las clases que los contenían (servlets del paquete control). Otro indicador fue el aspecto de tiempo de respuesta (profiling), quizás el medir

el tiempo que tarda en ejecutarse una función no es nada sorprendente pero cuando se mide el tiempo que tarda en ejecutarse un grupo de funciones desde una misma clase (aspecto) se ve la potencialidad de AOP, en el caso particular del prototipo de validación se midió el tiempo que tardaba en ejecutarse la inserción de: usuario, contenido y calificaciones, todo mediante la inserción de un aspecto en los métodos de las tres Beans de Gestión del paquete componentes: InsertarUsuario(), InsertarContenido() e InsertarCalificaciones().

RECOMENDACIONES

- Los aspectos tempranos son incumbencias transversales que se identifican en fases tempranas del ciclo de vida de desarrollo de software, incluyendo análisis de requisitos, análisis del dominio y finalmente diseño; por lo tanto para adaptar AOP se recomienda aplicar una metodología adecuada para diseñar aspectos desde las primeras fases de diseño (ingeniería de requisitos).

- AspectJ ha influenciado mucho en los trabajos existentes sobre AOP, sobretodo en las aproximaciones de modelado, ya que dependen mucho de su sintaxis; por tal motivo, se considera muy importante explorar otras herramientas.

- Las herramientas AOP disponibles para implementar aplicaciones Web son muy escasas o aún no están consolidadas, el caso de PHP orientado a aspectos; la recomendación para hacer desarrollo Web con AOP son las herramientas Java: Jboss AOP, Spring AOP.

- Para lograr aplicar la aproximación arquitectónica desarrollada en el trabajo se debe tener en cuenta el alcance de los aspectos dentro de la aplicación Web, es decir, una vez se defina que aspectos pueden ser aplicados hay que tener en cuenta los componentes que van a ser afectados por dichos aspectos sin que éstos entren en conflicto. También se debe tener en cuenta los métodos generales definidos en la descripción de cada uno de los aspectos y adaptarlos a la tecnología de implementación, independiente de esto se recomienda que sea una aproximación que soporte orientación a objetos.

TRABAJOS FUTUROS

- Se propone a partir de la base arquitectónica desarrollada diseñar una arquitectura formal utilizando herramientas especializadas en su diseño.

- Existe una herramienta UML muy interesante llamada UWE (UML - Based Web Engineering) que promueve el diseño de aplicaciones en tres perspectivas: modelo navegacional, modelo conceptual y modelo de presentación; aunque no soporta de manera total la introducción de aspectos resulta muy interesante trabajar con ella para modelar algunos aspectos que se escapan a las vistas tradicionales de UML; por ejemplo el aspecto de navegación.

- Resulta interesante poder unificar las aproximaciones UML que soportan aspectos en una sola extensión o perfil, tomando lo mejor de cada una y así definirlo de manera oficial, independiente de la tecnología y sin estar ajustados a AspectJ.

- Se podría mejorar el modelo definido con más incumbencias transversales para las aplicaciones Web, modelarlas e insertarlas en la capa de aspectos, definida en este trabajo.

REFERENCIAS

[1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier and J. Irwin, "*Aspect-Oriented Programming"*, Xerox Palo Alto Research Center, 1997.

[2] G. Kickzales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, J. Irwin. *"Aspect-Oriented Programming"*. Proceedings, European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241. Junio 1997.

[3] E. Baniassad, S. Clark. "Theme: An approach for aspect oriented analysis and design". International Conference on Software Engineering. 2.004.

[4] F. Asteasuain, B. Contreras. *"Programación Orientada a Aspectos: Análisis del Paradigma"*. Tesis de Licenciatura en Ciencias de la Computación. Universidad Nacional del Sur, noviembre de 2002.

[5] F. Matthijs, W. Joosen, B. Vanhaute, B. Robben, P. Verbaeten. "*Aspects should not die*." In European Conference on Object-Oriented Programming, Workshop on Aspect- Oriented Programming. 1997.

[6] N. Kicillof. (2005, Jul.)."Programación Orientada a Aspectos". MSDN Latinoamérica. http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art152.asp#authorbrief

[7] E. Yasser. Artículos en AOSD. (2008, Ene.). http://www.developer.com/design/article.php/3308941, http://aosd.net/

[8] S. Clarke. (2004, Abr.)."Diseño Orientado a Aspectos con Theme/UML".

[9] V. De Castro. E. Marcos. (2004, Jun.). "Representing WSDL with Extended UML". http://www.unab.edu.co/editorialunab/revistas/rcc/pdfs/r51_art1_r.pdf.

[10] JMX sitio oficial. (2008, Feb.). http://java.sun.com/jmx

[11] L. Dumont, D. Martínez, Ch. Metzner, N. Niño, Modelación de Aspectos con UML, Universidad Central de Venezuela, Escuela de Computación 2.005.

[12] AspectJ sitio oficial. (2008, Ene.). http://www.parc.xerox.com/aop/aspectj/

[13] Artículo Ingeniería del SW "Programación Orientada a Aspectos" Imagina Works. (2006, May.). http://www.imaginaworks.com/ .

[14] Artículo "Lenguaje Interpretado". (2006, Jul.). http://es.wikipedia.org/wiki/Lenguaje_interpretado

[15] R. Aaltman, A. Cyment, "SetPoint: Un enfoque semántico para la resolución de pointcuts en AOP". Tesis de Licenciatura. Departamento de Computación Facultad de Ciencias Exactas. Universidad de Buenos Aires. Noviembre 2.004.

[16] AOP/ST sitio oficial.(2008, Feb.). http://www.germany.net/teilnehmer/101,199268/

[17] A. Reina. "Visión General de la Programación Orientada a Aspectos". Departamento de Lenguajes y Sistemas Informáticos Universidad de Sevilla. Diciembre 2.005.

[18] I. Jacobson, G. Booch, J. Rumbaugh. "The Unified Software Development Process". Addison-Wesley. 1998.

[19] Sitio Oficial MFAOPHP. (2006, Ago.). http://www.mfaop.com/

[20] B. Saunders, J. Stamey, aoPHP Project, Dept. of Computer Science of Coastal Carolina University. (2006, Dic.). http://www.aophp.net

[21] Artículo sobre Middleware. (2006, Jul.). http://es.wikipedia.org/wiki/Middleware

[22] Syntactic analysis - Yacc & Parse Trees. (2007, Feb.). http://www.cs.man.ac.uk/~pjj/cs5031/ho/node5.html

[23] I. Krechetov, B. Tekinerdogan, A. Garcia, et Al. "Towards an Integrated Aspect-Oriented Modeling Approach for Software Architecture Design". AOSD 2.006

[24] M. Kandé, J. Kienzle, A. Strohmeier. "From AOP to UML: Towards an Aspect-Oriented Architectural Modeling Approach". Software Engineering Laboratory. Swiss Federal Institute of Technology Lausanne. (2007, May.). http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=545AD9958E98E966A5E79FDBF9FE6B03?doi=10.1.1.5.5015&rep=rep1&type=pdf

[25] Sitio Oficial de Spring Framework. (2007, Oct.). http://www.springframework.org/documentation

[26] P. Merson. “Using Aspect-Oriented Programming to Enforce Architecture”. Carnegie Mellon University. Software Engineering Institute. Septiembre 2.007.

[27] A. Rendón. “Proceso Unificado de desarrollo Lenguaje Unificado de Modelado”. Departamento de Telemática. Universidad del Cauca. Noviembre 2.001.

[28] M. García. “Tutorial JSP Java Server Pages”. (2007, Nov.). www.apl.jhu.edu/~hall/java/Servlet-Tutorial/

[29] A. Ordoñez. (2007, May.).“Introducción a JSP’s - Material Electiva Aplicaciones Web”. Departamento de Telemática. Universidad del Cauca.

[30] A. Ordoñez. (2007, May.). “Introducción a los Servlets - Material Electiva Aplicaciones Web”. Departamento de Telemática. Universidad del Cauca.

[31] I. Paez, C. Herrera. “Experimentación y evaluación del framework Jboss AOP”. Tesis Grado. Universidad EAFIT. Medellín. 2.007.

[32] Sitio Oficial de Jboss AOP. (2007, feb.). http://labs.jboss.com/jbossaop/

[33] F. Mostefaoui, J. Vachon. “Verification of Aspect-UML models using Alloy”. University of Montreal Quebec, Canada. Mayo 2.007.

[34] I. Groher, T. Baumgarth. ”Aspect-Orientation from Design to Code”. Siemens AG. Munich, Alemania. Agosto 2.007.

[35] E. Barra, G. Génova, J. Llorens. “An approach to Aspect Modelling with UML 2.0”. Departamento de Ciencias de la Computación. Universidad Carlos III de Madrid. Enero 2.007.

[36] A. Navasa, K. Palma, J.M. Murillo, Y. Eterovic. “Dos modelos arquitectónicos para el DSOA”. Universidad de Extremadura, Pontificia Universidad Católica de Chile. DSOA 2.004.

[37] Aspect Oriented Software Development (Tools for Developers), (2006, Sep.). http://www.aosd.net/wiki/index.php?title=Tools_for_Developers

[38] Artículo del Sitio Java en Castellano “Generar Código con XDoclet y el Plugin MyEclipse”, (2007, Ene.). http://www.programacion.net/java/articulo/jap_eclip_6/

[39] J. M. Nieto. “AspectJ en la Programación Orientada a Aspectos”. Escuela Técnica Superior de Ingeniería Informática. Universidad de Sevilla, España. Septiembre 2.007.

[40] M. Lorente, F. Asteausain, B. Contreras. “Programemos en AspectJ”. Versión 1.1 Septiembre 2.005.

[41] B. Haak, M. Díaz, C. Marcos, J. Pryor. “IDENTIFICACIÓN TEMPRANA DE ASPECTOS”. ISISTAN Instituto de Sistemas Tandil, Facultad de Ciencias Exactas, UNICEN. Argentina. Diciembre 2.007.

[42] AOSD Research Interest Group. Facultad de Ciencia y Tecnología. Universidade Nova de Lisboa. (2007, Ago.).Sitio Oficial: http://aosd.di.fct.unl.pt/aosd-group/

[43] J. Suzuki y Y. Yamamoto. “Extending UML with Aspects: Aspect Support in the Design Phase”. 3rd AOP workshop at ECOOP’99. Lisboa, Junio 1.999.

[44] M. Basch, A. Sanchez. “Incorporating Aspects Into the UML”. Department of Computer and Information Sciences. University of North Florida. Mayo 2.007.

[46] A. Cáceres. Centro de Investigación y de Estudios Avanzados – IPN. México D.F. 2.004.

[47] Artículo en TheServerSide.com. (2008, Abr.). “Implementing object caching with AOP”. http://www.theserverside.com/tt/articles/article.tss?l=ObjectCachingWithAOP

[48] C. Reynoso, N. Kicillof. “De lenguajes de descripción arquitectónica de software (ADL)”. Universidad de Buenos Aires. Marzo 2.004.

[49] P. Tarr, J. Ossher. “Workshop on Advanced Separation of Concerns in Software Engineering”. IEEE Computer Society. 2.001.

[50] Artículo sobre PHP. (2008, Jun.). http://es.wikipedia.org/wiki/.php

[51] Articulo “Análisis de rendimiento (Profiling) de aplicaciones Web con eclipse”. (2.007 Abr.). http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=ProfilingEclipse .

[52] A. Rashid, R. Chitchyan. “Persistente as an aspect”. Departamento de Computación. Universidad de Lancaster. 2.003