El Desarrollo de Software Orientado a Aspectos: Un Caso Práctico para un Sistema de Ayuda en Línea

Aspect-Oriented Software Development: A Practical Case for an On-line Help Desk System

Marta S. Tabares B. Ph.d.,[1] Germán H. Alferez Salinas MSc.,[2] Edward M. Alferez Salinas MSc.[3]
1. Escuela de Ingeniería de Antioquia, Colombia
2. Universidad de Montemorelos, México
3. Universidade Nova de Lisboa, Portugal
pfmstabare@eia.edu.co; harveyalferez@um.edu.mx; mauricio.alferez@di.fct.unl.pt

Recibido para revisión 15 de Marzo de 2008, Aceptado 19 de Mayo de 2008, Versión final 21 de Mayo de 2008

Resumen— El Desarrollo de Software Orientado a Aspectos (DSOA) provee un conjunto de enfoques para identificar, modularizar e implementar intereses o propiedades del sistema que pueden cruzar otros intereses del sistema. También busca mejorar el entendimiento de cada interés del sistema de forma clara y separada desde las primeras etapas del ciclo de vida de software. Este se orienta a la obtención de productos de software de calidad con partes más reutilizables y que evolucionen fácilmente en el tiempo. En este artículo, se presenta un caso de estudio para ilustrar la aplicación del DSOA desde etapas tempranas del desarrollo de software hasta la implementación. Diferentes enfoques orientados por aspectos se aplican para facilitar el manejo separado de intereses desde su identificación, representación en UML (análisis y el diseño), hasta su implementación en el lenguaje AspectJ.

Palabras Claves: Aspecto, Interés, Propiedad, Interés Transversales, AspectJ, Desarrollo de Software Orientado a Aspectos.

Abstract—Aspect-Oriented Software Development (AOSD) provides a set of approaches to identify, modularize and implement aspects or properties of the system that can crosscut other aspects. Also, it aims at improving the understanding of each feature of the system in a clear and separated way from early phases of the software development process. Thus, it is possible to obtain high quality software products that are easy to reuse and to evolve. In this paper, we present a case study in order to illustrate the application of AOSD from early development stages until the implementation. Several aspect-oriented techniques are applied to facilitate the separation of concerns and its representation in UML during analysis and design stages, as well as for its implementation in the AspectJ language.

Keywords: Aspect, Concerns, Crosscutting Concerns, AspectJ, Aspect-Oriented Software Development.

I. INTRODUCCIÓN

La Orientación a Aspectos (OA) surgió de necesidades muy concretas desde el nivel de programación. Define un mecanismo que ayuda a resolver problemas complementarios de código disperso (scattered) y enmarañado (tangled) que no se resuelven fácilmente usando paradigmas de desarrollo de software tradicionales como la orientación a objetos. Provee una unidad modular llamada Aspecto y un mecanismo de composición que permite entremezclar unidades modulares de comportamiento común con otras unidades modulares básicas del sistema [1].

OA se basa en el principio de "separation of concerns" (separación de intereses, asuntos o propiedades del sistema). Este principio se orienta hacia la descomposición del dominio del problema y de la solución, con el fin de reducir la complejidad, eliminar fallas de interpretación, y estructurar sistemas complejos a través de subsistemas, módulos o elementos simples de una forma más natural [2, 3]. La descomposición está orientada a la modularización del espacio del problema bajo una perspectiva dual, es decir: una, define los módulos principales de la solución (también llamados intereses base – base concerns en ingles), y la otra separa la funcionalidad transversal en módulos independientes (ortogonales) denominados intereses transversales (crosscutting concerns) o aspectos al nivel del diseño y la implementación. La composición está dirigida a cruzar (tejer) este tipo de intereses en diferentes módulos del núcleo del sistema.

El comportamiento cruzado puede corresponder a modelos, políticas, reglas de negocio, vistas de usuario, procesos de

Revista Avances en Sistemas e Informática, Vol.5 No.2, Junio de 2008, Medellín, ISSN 1657-7663

negocio, y atributos de calidad tales como auditoría (logging), seguridad, persistencias, rendimiento, etc. Por ejemplo, los métodos del componente de "seguridad" comúnmente se solapan en otros componentes o clases del sistema funcional base. Este se puede separar como un interés (concern), que cruza transversalmente a muchos otros intereses del sistema

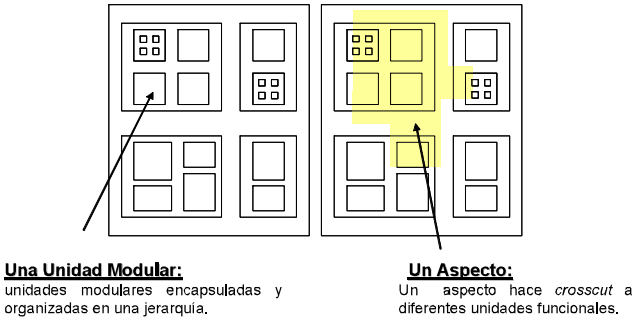

Figura 1. Descomposición modular vs. Aspectual [4]

DSOA[1] trata la separación de intereses a través del ciclo de vida de desarrollo. Particularmente, la identificación, modularización y especifican de los intereses se realiza en la Ingeniería de Requisitos Orientada a Aspectos (Aspect Oriented Requirements Engineering - AORE) , la cual provee la información necesaria para modelar los aspectos al nivel de la arquitectura, el diseño y la implementación [5, 6, 7, 8, 9, 10, 11, 12].

En este artículo, se presenta un caso de estudio que permite especificar las características y elementos necesarios para desarrollar software orientado a aspectos desde las primeras etapas del ciclo de vida hasta la implementación en el lenguaje AspectJ.

Este artículo se organiza de la siguiente forma: en la sección 2 se ilustran algunas de las características del DSOA.[2] En la sección 3 se desarrolla paso a paso un caso de estudio. Finalmente, en la sección 4 se concluye.

II. CARACTERÍSTICAS GENERALES DEL DESARROLLO DE SOFTWARE ORIENTADO A ASPECTOS

La Figura 2 ilustra la forma como los aspectos evolucionan a través de todo el ciclo de vida de desarrollo de software. Los aspectos se clasifican de la siguiente forma: (1) aspectos tempranos, los cuales se especifican desde los requisitos hasta la arquitectura; (2) aspectos intermedios, los cuales se especifican en estructuras aspectuales al nivel del diseño; (3) aspectos finales, los cuales se especifican en lenguajes de programación especializados como el AspectJ.

1 www.aosd.net
2 www.earlyaspects.net

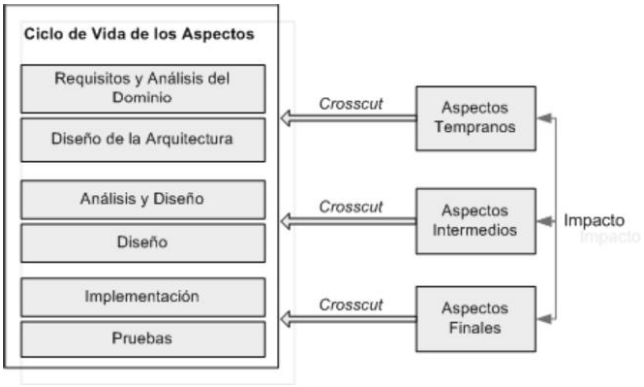

Figura 2. Ciclo de vida del Desarrollo de Software Orientado a Aspectos [10].

A. Aspectos Tempranos

AORE presenta diferentes enfoques que apoyan la elicitación y el análisis de requisitos de software [11]. Generalmente, los aspectos tempranos se basan en tres principios:

- Identificar, separar y componer intereses transversales.

- Soportar la negociación de conflictos (trade-off) causados por la identificación de intereses transversales, requisitos que se sobreponen unos con otros, o simplemente que están mal identificados o especificados.

- Negociar y tomar decisiones entre grupos de participantes (stakeholders).

Los enfoques en AORE presentan técnicas de descomposición/ composición de los requisitos funcionales, no funcionales y intereses transversales. Algunos enfoques, como Theme/Doc parten desde la descripción del problema en lenguaje natural para obtener artefactos de diseño agrupados por temas que posteriormente se tratan por medio de Theme/UML al nivel del diseño [7]. Otros, como AOSD/UC presenta los casos de uso como unidades de separación de intereses funcionales y no funcionales. Estos últimos son atributos de calidad que se representan en casos de uso de infraestructura. La composición se realiza por medio de un contenedor (paquete estereotipado) llamado use-case-slice, que permite especificar los aspectos en diferentes etapas del ciclo de vida [12].

B. Aspectos Intermedios

El Desarrollo de Software Orientado a Aspectos, inicialmente toma fuerza debido a la necesidad de expresar en un lenguaje de diseño, como UML, los elementos o estructura que ya era posible lograr al nivel de la implementación en AspectJ (u otros lenguajes orientados por aspectos). Hasta el momento, la comunidad OMG no ha reconocido a los Aspectos como un elemento que puedan ser representados y especificados en UML. Por esa razón, algunos enfoques han propuesto modificar el metamodelo UML tratando el aspecto como un clasificador [7, 13, 14]. Así, los aspectos en análisis y diseño normalmente se representan con clases y asociaciones estereotipadas (basadas en AspectJ).

Por ejemplo, en Theme/UML [7], los temas (themes) representan un elemento de diseño que agrupa una colección de estructuras (clases) y comportamiento (diagramas de secuencia) que representan una característica del sistema. Los temas se relacionan entre si, y su composición se especifica en relaciones de composición que identifican los elementos que se encuentran en solapamiento en diferentes modelos y especifican cómo los modelos se deben integrar.

Los aspectos intermedios también se definen al nivel de la arquitectura [15, 16, 17]. Enfoques como Aspectual Software Architecture Analysis Method (ASAAM) [17], presenta un mecanismo para tratar los intereses, al nivel de la arquitectura de diseño, que cruzan varios componentes arquitectónicos y no se pueden manejar fácilmente usando mecanismos de abstracción habituales.

C. Aspectos Finales

Kiczales define que: "un Aspecto es una unidad modular que se disemina por la estructura de otras unidades funcionales. Los Aspectos existen tanto en la etapa de diseño como en la de implementación. Un Aspecto de diseño es una unidad modular del diseño que se entremezcla en la estructura de otras partes del diseño. Un Aspecto de programa o de código es una unidad modular del programa que aparece en otras unidades modulares del programa". En otras palabras, un aspecto es una propiedad de un programa que no puede ser claramente encapsulada en un procedimiento generalizado (tal como un objeto, método, procedimiento o API) [1].

Varios lenguajes de programación se han creado o adaptado con nuevos componentes para desarrollar software orientado a aspectos. Algunos de ellos son: JPAL, D, COOL, RIDL, Aspecto, Aspecto, AspectS, AspectC++, MALAJ, HyperJ, etc. En este artículo presentamos las características generales de AspectJ ya que será el lenguaje de aplicación en el caso de estudio.

AspectJ [18], extiende a Java con soporte para dos tipos de implementación de corte trasversal. El primero, permite adicionar comportamiento (advices) en ciertos puntos (join points) bien definidos durante la ejecución de los programas - mecanismo de dynamic crosscutting -. El segundo, permite definir nuevas operaciones sobre tipos existentes - mecanismo de static crosscutting - porque afecta la declaración estática de tipos del programa.

AspectJ está basado en un grupo pequeño pero poderoso de constructores. Estos constructores son los siguientes:

* Join points: puntos bien definidos en la ejecución de un programa.

* Pointcuts: identifican conjuntos específicos de join points. Son filtros para aplicarse al Advice.

* Advices: Es una pieza de código que es aplicada a un pointcut. Son como métodos constructores usados para definir comportamiento adicional en el join point. Además declara cuándo es implementado dicho comportamiento (e.g. before, alter, around).

Cuando se desea conocer, por ejemplo, qué aspectos afectan una clase cuando se revisa el código fuente, es útil trabajar con el soporte de un IDE que permita entender cuales Aspectos afectan cualquier clase. Esto permite a los programadores de AspectJ beneficiarse de la modularización de los intereses transversales mientras mantienen acceso inmediato a los aspectos que afectan una clase [5].

Para la implementación de los aspectos es muy importante tener en cuenta el concepto de tejido, introducido por la OA. El tejedor (weaver), se encarga de llevar a cabo la mezcla entre los aspectos con la aplicación base tal como se muestra en la Figura 3 [19].

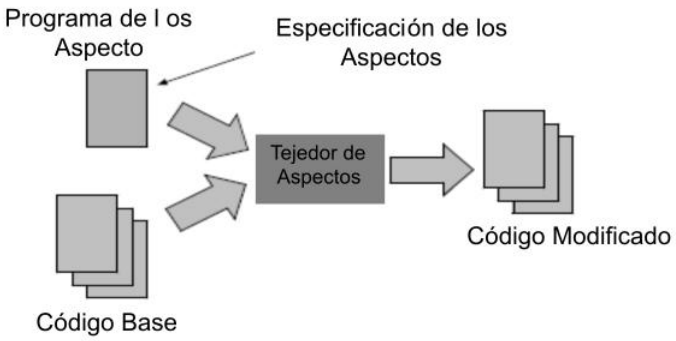

Figura 3. Proceso de tejido de Aspectos con el código base [19].

III. DESARROLLO DEL CASO DE ESTUDIO

El caso de estudio corresponde a un sistema de "Mesa de Ayuda". Este sistema está compuesto por tres módulos principales: Gestión del Sistema, Gestión de Solicitudes y Gestión de Técnicos. No obstante, en el estudio tomamos los dos primeros módulos debido a que son los más complejos y con ellos se puede entender la potencia de la Orientación a Aspectos. Además, este sistema está diseñado en tres capas: presentación (JSP's), dominio y servicios técnicos (clases de Java).

A. Descripción del caso de estudio

El problema corresponde a un desarrollo orientado a objetos que se localiza en el Departamento de Mantenimiento de una universidad [20, 22]. Su funcionalidad corresponde a la atención en línea de solicitudes de servicios de mantenimiento de edificios. El usuario afectado por el daño debe diligenciar una solicitud en la cual hace una descripción del problema. Dicha solicitud podrá ser ingresada a cualquier hora del día y cualquier día de la semana, en cualquier terminal de cómputo de la empresa. Un empleado del área de mantenimiento la recibe y valida; si está correcta, programa los técnicos para la atención del servicio en una agenda de servicios.

Para desarrollar el estudio se consideraron los siguientes pasos:

- Identificación de Aspectos tempranos: descripción de requisitos y intereses dentro del dominio del problema.

Revista Avances en Sistemas e Informática, Vol.5 No.2, Junio de 2008, Medellín, ISSN 1657-7663

- Modelado de los intereses en el análisis y diseño
- Especificación de los aspectos en AspectJ.

B. Identificando y Especificando Aspectos Tempranos

A partir de la descripción general del problema [20], mostramos algunos de los requisitos funcionales y no funcionales que permiten desarrollar la solución orientada a aspectos.

Requisitos:

1. Diligenciar solicitudes (solo por usuarios matriculados en el sistema).
2. Evaluar la solicitud por parte del técnico
3. Rechazar solicitudes
4. Asignar los servicios que requiere la solicitud
5. Ejecutar los servicios programados por solicitud diligenciando una plantilla de servicios.
6. Contratar servicios con empresas externas (esto requiere autorización del jefe de área y del departamento financiero).
7. Hacer seguimiento del flujo de trabajo de una solicitud de servicio.
8. Escalar servicios
9. Reportar estadísticas de servicio.
10. Proveer buen tiempo de respuesta cuando se hace una transacción sobre el sistema.
11. Soportar muchos usuarios concurrentes en cualquier operación.
12. Notificar a los solicitantes cuando una solicitud cambia de estado.
13. Costear el trámite de la gestión de los servicios internos y externos.

Intereses identificados:

Solicitudes, Servicios, Gestión de usuarios, Seguridad, Disponibilidad, Compatibilidad, Multiacceso, Soporte al sistema, Sistemas Externos, Estadísticas de Gestión, Contratos, Notificación, Costos Operacionales.

En la Tabla 1 se cruzan algunos intereses con unidades requisitos funcionales y no funcionales del sistema. Así, será posible identificar intereses de corte transversal (crosscutting concerns) y a su vez validar la correcta especificación de los requisitos asociados a los intereses. Para especificar los aspectos tempranos, utilizamos la técnica de elicitación de Casos de Uso [12]. En la Figura 4, se ilustra un modelo de casos de uso que corresponde de forma general a representación los anteriores intereses/requisitos.

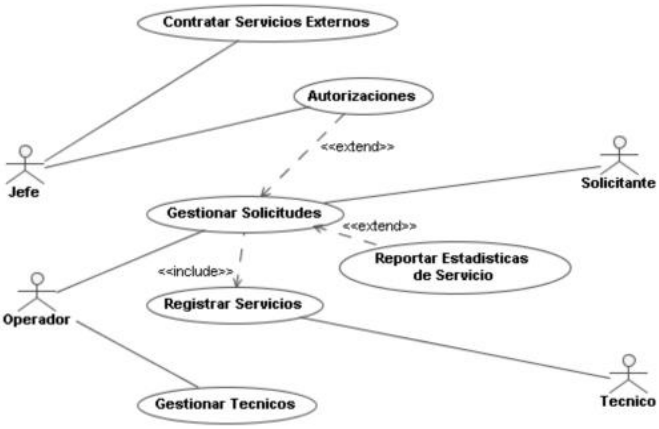

Figura 4. Modelo de Casos de Uso para el sistema de Ayuda en Línea.

Tabla 1. Intereses vs. Requisitos

	Intereses (*Concerns*)			
Requisitos funcionales y no funcionales	**Solicitudes**	**Servicios**	**Costos Operacionales**	**Notificar**
Diligenciar solicitud	X			X
Evaluar solicitud	X			
Asignar servicios		X		X
Contratar servicios		X		
Autorizar solicitudes	X	X		X
Rechazar solicitudes	X	X		X
Ejecutar servicios		X		
Notificar cambio de Estado	X	X		X
Costear tramite servicio	X	X	X	X
Escalar servicio		X		
Proveer buen tiempo de respuesta	X	X	X	
Soportar usuarios concurrentes	X	X		
Costear trámite de servicios	x	x	x	

Descripción de los Aspectos que serán utilizadas en las fases posteriores del desarrollo:

AspectoNotificar: Este aspecto le notifica al Solicitante a través de un e-mail las decisiones que se toman con respecto a su Solicitud y sus Servicios: rechazo o escalamiento de la solicitud o del servicio, o contratación de un servicio.

AspectoCostosOperacionales: Este aspecto costea las siguientes operaciones: ingresar solicitud, ingresar servicio, escalar solicitud, escalar servicio, rechazar solicitud, rechazar servicio, y contratar servicio.

AspectoMedirTiempos: Este aspecto toma el tiempo en milisegundos de las siguientes operaciones con el fin de verificar que los usuarios tienen un alto nivel de servicio: registrar solicitud, ingresar servicio, rechazar solicitud, rechazar servicio, escalar solicitud, escalar servicio, y contratar servicio.

Usamos la aproximación AOSD/UC [12] para representar los intereses de infraestructura por medio del caso de uso <Perform Transaction>. Por medio de este caso de uso, los flujos alternos se pueden representar en casos de uso de extensión (ver Figura 5). En el caso de uso <Perform Transaction> se deben identificar los puntos de extensión y definir los pointcuts por cada caso de infraestructura que extiende la función básica de una transacción.

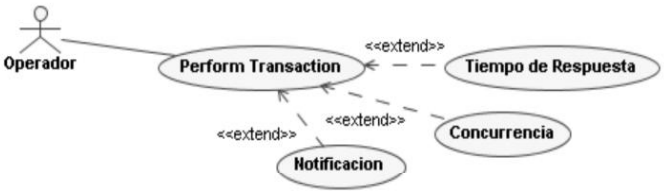

Figura 5. Tratamiento de los atributos de calidad bajo la aproximación AOSD/UC [12].

C. Modelado de Intereses en el Análisis y Diseño

En esta fase del desarrollo de software orientado a aspectos, los intereses son representados en elementos de modelo dispuestos por UML para el diseño. Específicamente, como se explicó en la sección 2.2 los aspectos son representados en elementos estereotipados. Para este caso de estudio, los intereses (aspectos) son especificados desde los diagramas de secuencia basados en [14]. Estos permiten ver el flujo de las operaciones que los flujos básicos de los casos de uso proveen. Además, permiten ver la forma como éstas pueden ser intervenidas por las operaciones que determinan los flujos de los casos de uso de extensión. En la Figura 6 se muestra una parte del diagrama de secuencia "Escalar Solicitud".

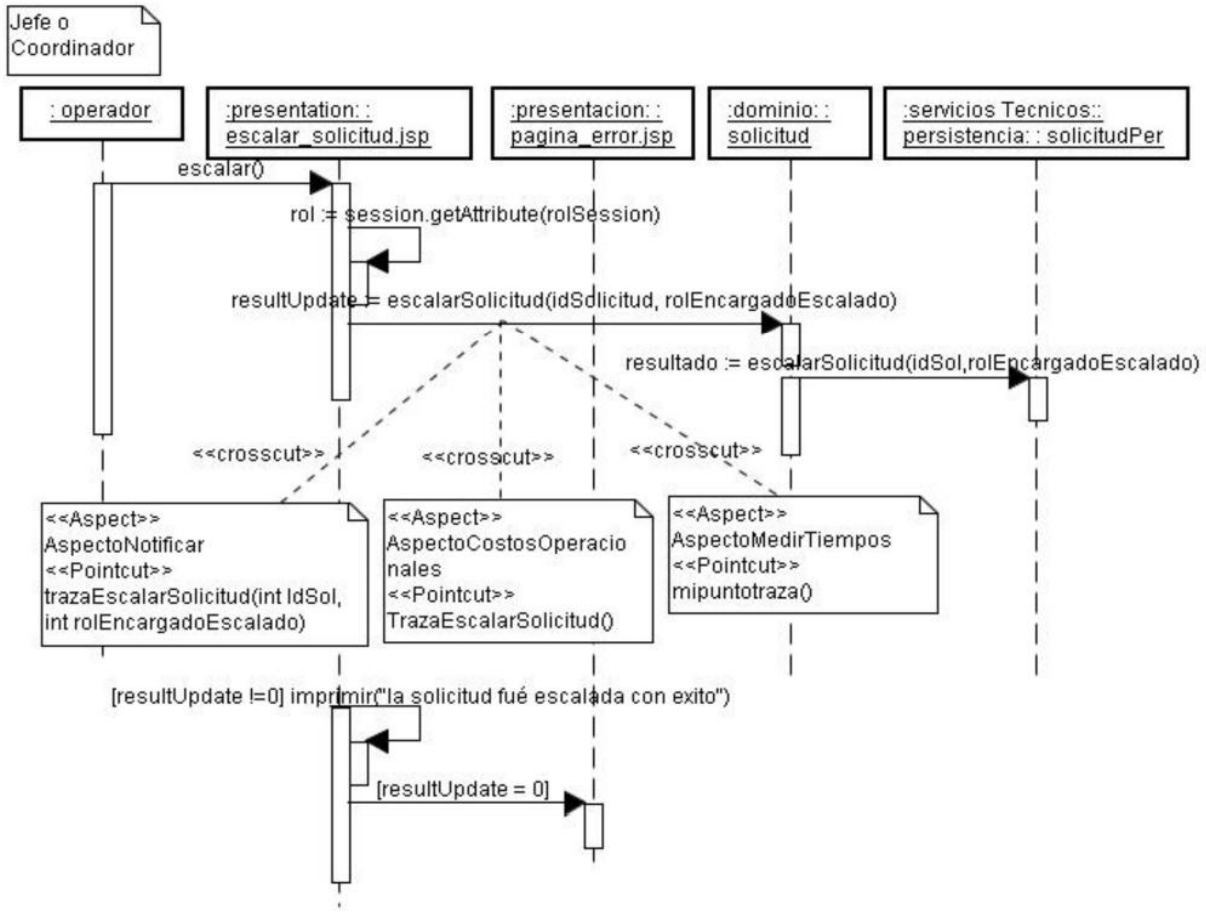

Figura 6. Parte del diagrama secuencial Escalar Solicitud utilizando los intereses (aspectos) Notificar, CostosOperacionales y MedirTiempos.

Revista Avances en Sistemas e Informática, Vol.5 No.2, Junio de 2008, Medellín, ISSN 1657-7663

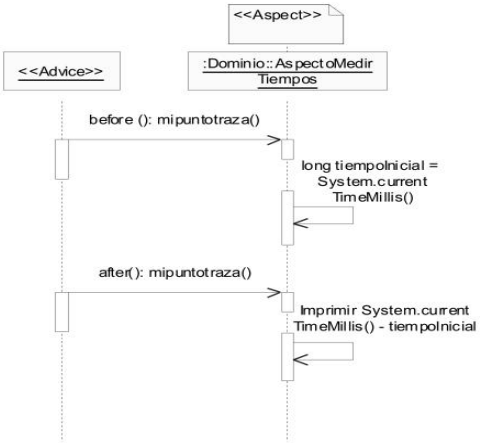

Figura 7. Diagrama secuencial del pointcut AspectoMedirTiempos.

En la Figura 8 se muestra un trozo del diagrama de clases de diseño con aspectos. Se puede observar al aspecto AspectoMedirTiempos que se define en la capa de Dominio cruzando transversalmente las clases Dominio y Solicitud que se encuentran en la misma capa de la arquitectura.

Figura 8. Parte del diagrama de clases de diseño del interés MedirTiempos.

D. Especificación de los aspectos en AspectJ.

Para la implementación de los intereses analizados en el caso de estudio, se utilizaron las siguientes herramientas:

- Plataforma Eclipse.
- Plug-in MyEclipse (http://www.myeclipseide.com): Permite trabajar aplicaciones J2EE en el marco de trabajo de Eclipse.
- Plug-in AJDT (AspectJ Development Tools).
- Apache Tomcat 5.0.27 que es un contenedor de servlets y JSPs (servidor de aplicaciones).
- MS Access 2002 en el que se creó la base de datos.

En la Figura 9 se presenta a manera de ejemplo el código del método "rechazarSolicitud" de la clase Solicitud, el cual sin la implementación de los aspectos presenta problemas de scattering y tangling. Se puede observar que el código dentro del cuadro con líneas discontinuas está relacionado con el Aspecto Funcional de notificación, el código dentro del cuadro con doble línea está relacionado con el Aspecto Funcional del cálculo de los costos operacionales, y los códigos en los cuadros con línea sencilla están relacionados con el Aspecto no Funcional de medición de tiempos de ejecución.

```
public int rechazarSolicitud(int idSoli,
                             int idSust){
int resultado = 0;
  long tiempoInicial=0;
  System.out.println("--> rechazarSolicitud");
  tiempoInicial = System.currentTimeMillis();
  resultado = SolicitudPer.rechazarSol(idSoli,
  idSust);
  correo.notificar("Rechazo de la solicitud ",
  idSoli, idSust, 0);
  CostosOperacionales.costosRechazarSolicitud()
  System.out.println("<-- rechazarSolicitud"+
                     " duración: " +
  (System.currentTimeMillis() - tiempoInicial ));
        return resultado;
}
```

Figura 9. Señalamiento de las líneas de código en el Método rechazarSolicitud

En la Figura 10 se presenta en AspectJ el aspecto "AspectoMedirTiempos" el cual a través del pointcut mipuntotraza interviene a las operaciones ingresar solicitud, rechazar solicitud, ingresar servicio, etc, de las clases Solicitud y Servicios.

```
package dominio;
import java.util.*;
import org.aspectj.lang.*;
public aspect AspectoMedirTiempos {
private long tiempoInicial = 0;
        pointcut mipuntotraza():
        (execution(* Solicitud.ingresarSol(..)))||
        (execution(* Solicitud.rechazarSolicitud(..)))||
        (execution(* Solicitud.escalarSolicitud(..)))||
        (execution(* Servicio.ingresarServicio(..)))||
        (execution(* Servicio.rechazarServicio(..)))||
        (execution(* Servicio.contratarServicio(..)))||
        (execution(* Servicio.escalarServicio(..)));

        before (): mipuntotraza()
        {
Signature sig = thisJoinPointStaticPart.getSignature();
        System.out.println("--> " +
          sig.getDeclaringType().getName() + "."+
sig.getName());
        tiempoInicial = System.currentTimeMillis();
        }
        after(): mipuntotraza()
        {
Signature sig = thisJoinPointStaticPart.getSignature();
        System.out.println("<-- " +
          sig.getDeclaringType().getName() + "."+
 sig.getName()+ " duración: " +
 (System.currentTimeMillis() - tiempoInicial )+
" milisegundos.");
        }
}
```

Figura 10. Implementación del Aspecto "AspectoMedirTiempos"

Gracias a los aspectos que se implementaron en el caso práctico, se separaron los intereses de una manera que permite entre otras cosas, un mejor entendimiento y reutilización del código. Basta con observar el código de la Figura 11 y compararlo con el de la Figura 9 para cerciorarse de las mejoras.

```
public int rechazarSolicitud(int idSoli, int idSust){
        int resultado = 0;
resultado = SolicitudPer.rechazarSol(idSoli, idSust);
        return resultado;
}
```

Figura 11. Método "rechazarSolicitud" al ser implementado bajo la Orientación a Aspectos

La Figura 12 ilustra al método "rechazarSolicitud" tejido con el Aspecto "AspectoMedirTiempos". Los advices se activan antes y después de la ejecución del método. Los rectángulos en esta figura indican las locaciones que representan join ponts durante el tiempo de ejecución.

```
public int rechazarSolicitud(int idSoli, int idSust){
 advice(...);
int resultado = 0;
long tiempoInicial=0;
...
return resultado;
 advice(...);
}
```

Figura 12. Método "rechazarSolicitud" tejido con el Aspecto "AspectoMedirTiempos"

IV. CONCLUSIONES

El DSOA permite extender un sistema de software existente, iteración por iteración, durante el tiempo de vida del sistema al manejar adecuadamente los intereses transversales, ayudando a alcanzar una mejor calidad del software.

Es importante tener en cuenta la Orientación a Aspectos desde las etapas tempranas del ciclo de vida de desarrollo de software ya que se puede anticipar el razonamiento acerca del tratamiento de aspectos (separación de concerns y manejo de crosscutting concerns). En otras palabras, esto hace posible el entendimiento de un sistema de intereses a través de los requisitos, análisis y modelos de diseño, en lugar de demandar que su entendimiento sólo dependa del análisis de artefactos de implementación.

La introducción de aspectos en la etapa de especificación de requisitos permite realizar un tratamiento sistemático relacionado con el DSOA desde el principio del ciclo de vida que sirve para guiar las técnicas de la Programación Orientada a Aspectos (POA) al final del ciclo de vida. Estas buenas prácticas de desarrollo promueven la trazabilidad de los requisitos e intereses transversales a través de todas las etapas del ciclo de vida, mejorando así el mantenimiento y la evolución del sistema.

En la etapa de implementación, se encontró que la utilización del plug-in AJDT ayuda al desarrollador en obtener un mejor manejo de los aspectos en cuanto a la navegabilidad y la visualización de sus estructuras. Junto con el plug-in MyEclipse facilita la creación de arquitecturas multinivel J2EE que utilizan aspectos. Por otro lado, en la experimentación se comprobó que la OA hace posible programar los intereses transversales en una forma modular y así alcanzar sus beneficios: código más simple y más fácil de desarrollar y mantener, y que tiene gran potencial para la reutilización. Además, se cumplieron los dos objetivos principales que propone la POA: separar los intereses, y minimizar las dependencias entre ellos. El primer objetivo se obtuvo al modularizar cada interés durante el proceso de desarrollo; el segundo se logró al minimizar el acoplamiento entre los distintos elementos. Finalmente, se observaron las siguientes características positivas:

* Un código menos enmarañado, más natural y más reducido.

* Una mayor facilidad para razonar sobre los intereses separados con una dependencia mínima de otros intereses.

* Facilidad para la depuración y las modificaciones en el código.

* Un código más reutilizable que se puede acoplar y desacoplar cuando sea necesario.

V. TRABAJO FUTURO

Nuestro trabajo actualmente está siendo extendido para incluir trazabilidad y técnicas de desarrollo y generación automática de modelos. Trazabilidad de intereses entre cada una de las etapas del desarrollo de software va a ayudar a crear software más fácilmente mantenible y con soporte para análisis del impacto de cambios. Así, los desarrolladores podrán tomar decisiones bien informadas al momento de modificar artefactos de software o para medir el impacto de la introducción de nuevos concerns en el sistema a largo de todas las etapas del desarrollo de software.

REFERENCIAS

[1] Kiczales G.; Hilsdale, E.; Hugunin, J.; Kersten, M.; Palm, J. y Griswold, W.G., Oct. 2001. Getting started with AspectJ. En : Communications of the ACM, Vol. 44, n. 10, pp. 59-65.

[2] Dijkstra E.W., 1976. A Discipline of Programming, NJ: Prentice Hall.

[3] Parnas D., On the criteria to be used in decomposing systems into modules, 1972. En : Communications of the ACM, pp. 1053-1058.

[4] Czarnecki K., Eisenecker U.W., Generative Programming Methods, Tools, and Applications, 2000. Addison-Wesley.

[5] Kiczales G.; Hilsdale E.; Hugunin J.; Kersten M.; Palm J. y Griswold W. G, 2001. Getting started with AspectJ. En : Communications of the ACM, Vol. 44, n. 10, pp. 59-65.

[6] Tarr P.; Ossher H.; Sutton S.M. Jr., 1999. N Degrees of separation: multidimensional separation of concerns. En: Proceedings of the 21st International Conference on Software Engineering. pp. 107-119.

[7] Clarke S.; Baniassad, E., 2005. Aspect-Oriented Analysis

and Design. The Theme Approach. Addison-Wesley, Object Technology Series.
[8] Rashid A.; Moreira A. and Araújo, J., 2003. Modularisation and composition of aspectual Requirements. En: Proceedings of the 2nd international conference on Aspect-oriented software development, ACM, pp. 11-20.
[10] Tekinerdogan B., Moreira A., Araújo J., Clements P. (Eds), 2004 . Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design. En: Workshop Proceedings Lancaster, UK. http://early-aspects.net/,
[11] Chitchyan R.; Rashid A.; Sawyer P.; Bakker J.; Pinto M.; Garcia A.; Tekinerdogan B.; Clarke S.; Jackson A., May 2005. Survey of Aspect-Oriented Analysis and Design Approaches, AOSD-Europe-ULANC-9, AOSD-EUROPE network of excellence.
[12] Jacobson I.; Ng P.W., 2005. Aspect-Oriented Software Development with Use Cases, Redwood City: Addison-Wesley.
[13] Suzuki J.; Yamamoto Y., 1999. Extending UML with aspects: aspect support in the design phase. En: Proceedings of the third ECOOP Aspect-Oriented Programming Workshop.
[14] Stein D.; Hanenberg S.; Unland R., 2002. A UML-based aspect-oriented design notation for AspectJ. En : Proceedings of the 1st international conference on Aspect-oriented software development. pp. 106-112.
[15] Kande M.M., 2003. A Concern-oriented Approach to Software Architecture. PhD thesis, Ecole Polytechnique Federal de Lausanne, n. 2796.
[16] Pinto M.; Fuentes L. y Troya J.M., 2003. DAOP-ADL: An Architecture Description Language for Dynamic Component and Aspect-Based Development, Presented at International Conference on GPCE, Erfurt, Germany.
[17] Tekinerdogan B., 2004. ASAAM: Aspectual software architecture analysis method. Presented at WICSA 4th Working IEEE/IFIP Conference on Software Architecture.
[18] Kiselev I., 2003. Aspect-Oriented Programming with AspectJ. Editorial Indianapolis : Sams.
[19] Birrer I.; Chevalley P.; Pasetti A.; y Rohlik O., 2004. An Aspect Weaver for Qualifiable Applications". En : Proceedings of the 15-th Data Systems in Aerospace (DASIA) Conference.
[20] Pineda O.; Zapata M., 2004. Definición, análisis y diseño para el software de servicios generales UNAL Sede Medellín. Tesis de grado Ingeniería de Sistemas, Facultad de Minas, Universidad Nacional de Colombia (Medellín, Colombia).
[21] Masuhara H.; Kiczales, G.; y Dutchyn C.A, 2003. Compilation and Optimization Model for Aspect-Oriented Programs. En : Proceedings of Compiler Construction (CC2003), LNCS 2622, Springer-Verlag, pp. 46-60.
[22] Alférez E.M.; Alférez G.H. y Tabares, M.S.(a), 2004. Análisis de un sistema de información bajo la aproximación de la orientación a aspectos. Caso práctico: manejo de solicitudes en la mesa de ayuda de servicios generales de una universidad. Trabajo de grado Ingeniería de Sistemas. Facultad de Ingeniería, Universidad EAFIT (Medellín, Colombia).

Marta Silvia Tabares Betancur es Ingeniero de Sistemas y MSc en Ingeniería Informática de la Universidad EAFIT de Medellín, Colombia. En la actualidad es candidata a Doctor (PhD(c)) en Ingeniería de Sistemas de la Universidad Nacional de Colombia, Medellín. Además, está dedicada a la investigación y la docencia como profesora asociada de la Escuela de Ingeniería de Antioquia, Colombia (EIA), y lidera los proyectos de investigación y asesoría del grupo de investigación GIISEIA en Ingeniería de Software de la EIA. Sus intereses de investigación incluyen: Requirement Engineering, Requirement Traceability, Model-Driven Development, Aspect-Oriented Software Development, y Software Architectures.

Mauricio Alférez es un estudiante de PhD in Computer Science/Informatics de la Faculdade de Ciências e Tecnologia de La Universidade Nova de Lisboa, Portugal, e investigador del proyecto de la Unión Europea para el mejoramiento de técnicas de desarrollo para Líneas de Producto de Software – AMPLE. En 2004 se graduó en Ingeniería de Sistemas en la Universidad EAFIT. Desde entonces ha trabajado tanto en la industria como en la investigación para empresas como Universidad EAFIT, Colombia; Sistema Metro de Medellín (desarrollo del Sistema Portátil de Diagnóstico), Colombia; Termopaipa Power Plant - STEAG A.G, Colombia; y West Indies Union Conference, Jamaica. Sus intereses en investigación son: Model-Driven Software Development, Aspect-Oriented Software Development, y Software Product Lines.

Germán Harvey Alférez Salinas enseña a nivel de pregrado y de maestría en la Facultad de Ingeniería y Tecnología, Universidad de Montemorelos, México, y como investigador asociado en Mission College, Tailandia. En 2008, se graduó con honores en el Master of Science in Information and Communication Technology en Assumption University, Tailandia. En 2004 se graduó en Ingeniería de Sistemas en la Universidad EAFIT, Colombia. Entre sus trabajos más significativos se encuentra el ser coordinador del Computer Information Systems Department, Faculty of Business Administration en Mission College y profesor adjunto en la maestría de educación en Avondale College, Australia. Sus intereses de investigación son: Software Product Lines, Model-Driven Software Development, Aspect-Oriented Software Development y Software Architectures.