Collecting Metrics for the Development of Aspect Oriented Programming

Recolectando métricas para el desarrollo de la programación orientada a aspectos

Ing. Adriana Iglesias, Ms.
Barranquilla-Colombia
aiglesias3@unisimonbolivar.edu.co

Ing. Ricardo Llano Bravo
Barranquilla-Colombia
castlellanos@gmail.com

Keywords:
AOP, Aspect, Component, class, Woven class, Weave.

Abstract
The Aspect Oriented programming is a programming style that the main objective is to achieve adequate modularization of the concepts involved in an application, this result in the achievement of separation between the functional requirements of the non-functional, to get a better understanding of the concepts, eliminates dispersion of code and makes the implementations more readable, adaptable and reusable. Also provides a framework that allows the programmer to clearly separate components and aspects through mechanisms to allow abstracting and composing them to produce the overall system.

Palabras clave:
POA, Aspecto, Componente, Clase, Clase Tejida, Tejido.

La Programación Orientada a Aspectos es un estilo de programación cuyo principal objetivo es lograr una adecuada modularización de los conceptos involucrados en una aplicación, esto se traduce en lograr la separación entre los requerimientos funcionales de los no funcionales, para obtener un mejor entendimiento de los conceptos, eliminando la dispersión del código y haciendo que las implementaciones resulten más comprensibles, adaptables y reutilizable. A su vez proporciona un marco de trabajo que permite al programador separar claramente componentes y aspectos a través de mecanismos que hagan posible abstraerlos y componerlos para producir el sistema global.

INTRODUCCION

Este estudio del estado del arte es una descripción conceptual acerca de la Programación Orientada a Aspectos y a su vez proporciona información detallada sobre las métricas y conceptos que se derivan de este paradigma de programación, se toma como eje de la revisión los conflictos entre aspectos.

Las nuevas abstracciones en el desarrollo de software orientado a aspectos evitan la mezcla de diversos intereses y previene la separación de una funcionalidad en varios módulos, para lograr esto se establecen técnicas de modelado, análisis y diseño orientadas a aspectos. De manera general, muchos estudiosos en el tema han presentado investigaciones cuyo objeto es identificar aspectos en etapas de ingeniería de software anteriormente mencionadas.

Se demarcaran de igual forma las métricas que hasta el momento han surgido y/o se han propuesto por distintos investigadores, las cuales nos ayudaran a comprender aún más el desarrollo de la programación orientada a aspectos.

Se han propuesto diversas métricas para el desarrollo y medición de las propiedades de aplicaciones orientada a aspectos, pero muchas de éstas han sido cuestionadas. Todas éstas métricas han sido desarrolladas por diferentes autores, pero bajo el estudio que se ha realizado enseñaremos las métricas que más sobresalen y han sido aceptadas por la mayoría de los investigadores e ingenieros, es decir son métricas ya estipuladas de las cuales aun se encuentran en estudio y desarrollo.

La finalidad de éste articulo será identificar y definir todas esas métricas, dar un avance desde el punto de vista de diferentes autores, el proceso de cómo han ido evolucionando y como han sido aceptadas dentro de éste paradigma.

Principalmente entraremos a los distintos conceptos que se han adoptado para la programación orientada a aspectos y a partir de ahí se definirán las distintas métricas propuestas por los distintos autores y como estas han afectado el desarrollo del paradigma que se está planteando.

DEFINICIÓN DE PROGRAMACIÓN ORIENTADA A ASPECTOS

Cuando se habla de la Programación Orientada a Aspectos -POA-, es necesario definir lo que es un componente y un aspecto, en 1997 GregorKickzales explica que “un componente puede encapsularse claramente dentro de un procedimiento generalizado. Los componentes son unidades de descomposición funcional del sistema”. Además GregorKickzales define un aspecto de la siguiente manera, “no puede encapsularse claramente en un procedimiento generalizado, Suelen ser propiedades que afectan al rendimiento o a la semántica de los componentes” [1].

Por otro lado un estudio realizado por la universidad del Bío – Bío, define la Programación Orientada a Aspectos POA “como un estilo de programación cuyo principal objetivo es lograr una adecuada modularización de los conceptos involucrados en una aplicación” [2], esto ayuda a que el sistema desarrollado sea entendible para el programador, y se pueda hacer la reutilización de código, además es importante este concepto ya que ayuda evidenciar cuales son los posibles requerimiento funcionales del sistema y poderlos tener en cuenta a la hora de la implementación del mismo.

Una investigación que se realizo en la Universidad Nacional de San Luis argumenta que la POA define a una clase como, “la unidad básica que encapsula el comportamiento del sistema a través de las distintas funcionalidades y no contiene el comportamiento de los aspectos que la afectan” [3]. A esto se asocia el término de clase tejida en el cual “la estructura de dicha clase depende del tejedor de aspectos y del lenguaje de programación que se haya utilizado para generarla. Una clase contiene un punto de enlace los cuales son una clase especial de interfaz entre los aspectos” [4].

DEFINIENDO CONCEPTOS

Basándose en las definiciones anteriores se da lugar a la definición o el objetivo principal de la Programación Orientada a Aspectos –POA- GregorKickzales sustenta la siguiente definición “proporcionar un marco de trabajo que permita al programador separar claramente componentes y aspectos a través de mecanismos que hagan posible abstraerlos y componerlos para producir el sistema global.” [1]. Además un aspecto se define como la unidad modular la cual tiene como función

dispersarse por unidades funcionales del sistema, muchas veces ocurre cuando estamos diseñando el sistema o cuando se está en la etapa de implementación.

De la POA se derivan diferentes conceptos básicos tales como punto de cruce o de unión, este es el punto de ejecución dentro del sistema en el cual un aspecto puede ser invocado o llamado por medio de algún método del cual requiera. Seguidamente aparece el concepto de Consejo o Advice, es la implementación que se hace al aspecto definiendo cual va a ser su funcionalidad en el sistema. Los puntos de corte definen los Concejo que se aplicaran a cada punto de cruce. "Se especifica mediante expresiones regulares o mediante patrones de nombres (de clases, métodos o campos), e incluso dinámicamente en tiempo de ejecución según el valor de ciertos parámetros. De aquí se desglosa la introducción la cual permite añadir métodos o atributos a clases ya existentes" [5], a lo anterior se asocia el termino común de reutilización de código existente, esto ayuda mucho al programador ya que se asocia a la teoría de no inventar cuando se puede reutilizar, esta teoría consiste que si tienes código o información que te sirve para que inventar algo que ya está hecho y que solo se puede reutilizar, adaptándolo a las necesidades que este requiera.

"Además está el destinatario es la clase aconsejada, la clase que es objeto de un consejo. Sin POA, esta clase debería contener su lógica, además de la lógica del aspecto. Otro término relacionado resultante es el objeto creado después de aplicar el Consejo al Objeto Destinatario. El resto de la aplicación únicamente tendrá que soportar al Objeto Destinatario (pre-AOP) y no al Objeto Resultante (post-AOP)"[6]. Y por ultimo tenemos el tejido "es el proceso de aplicar Aspectos a los Objetos Destinatarios para crear los nuevos Objetos Resultantes en los especificados Puntos de Cruce. Este proceso puede ocurrir a lo largo del ciclo de vida del Objeto Destinatario:

- Aspectos en Tiempo de Compilación, que necesita un compilador especial.
- Aspectos en Tiempo de Carga, los Aspectos se implementan cuando el Objeto Destinatario es cargado. Requiere un ClassLoader especial.
- Aspectos en Tiempo de Ejecución" [7].

METRICAS

Existen métricas las cuales intentan medir la complejidad del software para predecir el costo total del proyecto, la evaluación y la efectividad del diseño. Estas ayudan a mantener la vida útil del software para que así perdure mucho más tiempo.

Existe la Métrica LDCF (Líneas De Código Fuente) esta métrica es utilizada para definir cuál es el tamaño del software por medio de las líneas del código fuente. La anterior utilizada para predecir cuál será el esfuerzo que se necesita para desarrollar el programa, además permite saber cuál será la productividad al momento de programar o cuando se hace entrega del producto software. De esta métrica se derivan dos tipos de medidas las Líneas De Código Fuente Físicas y Líneas De Código Fuente Lógicas, dentro de las Líneas De Código Fuente Físicas se encuentras todas aquellas líneas donde se comenta el programa. Una contrapartida que posee Líneas De Código Fuente físicas, es que sensitiva al formato de tipo texto. Sin embargo, las Líneas De Código Fuente lógicas se basan en cómo se declaran las variables, instancias o llamadas de métodos, clases, etc. En el código fuente.

Las métricas utilizadas para la evaluación de sistemas orientados a objetos son aplicables a los sistemas orientados a aspectos, como es el caso de las seis métricas C&K definidas por Chidamber y Kemerer:

Tabla 1. Métricas de la Programación Orientada a Aspectos [8] [9] [10]

METRICA	**DESCRIPCION**
Peso de los Métodos por Clase [8]	Cuenta la cantidad de métodos que componen una clase. Cuando las clases contienen muchos métodos tienen mayores probabilidades de estar muy atadas a una aplicación, limitando su posibilidad de reúso, a su vez afecta a las clases derivadas, dado que heredan los métodos de la clase base.
Profundidad de árbol de Herencia [8]	Cuanto más profunda sea una clase en la jerarquía, tendrá más métodos heredados, haciéndola más compleja. Glasberg [9] indica "las clases más propensas a contener errores son las que están en la mitad del árbol"
Número de Hijos [8]	Miden la amplitud de la jerarquía de árbol, mientras que

	DIT mide la profundidad.
Acoplamiento entre objetos (CBO) [8]	Dos clases están acopladas cuando los métodos declarados en una clase utilizan métodos o variables de instancia definidas en otra clase. Un CBO alto es propenso a fallos, por lo que es necesario realizar pruebas rigurosas. Según HouariSahraoui "un CBO mayor a 14 es demasiado alto" [10].
Respuesta por Clase (RPC) [8]	RPC es el número de métodos locales de una clase más el número de métodos llamados por dichos métodos locales. RFC incrementa ante la presencia de aspectos.
Métrica de falta de cohesión (LCOM) [8]	Miden cuán bien los métodos de una clase están relacionados unos con otros.

Se dice que cuando la clase es muy cohesiva esta tiende a poseer un alto nivel de acoplamiento con los métodos de la clase, es decir las relaciones que se tienen entre sí al momento de ejecutar el programa, como lo es la llamada o invocación de algún método o clase en especifico. Por lo tanto cuando se tiene una baja cohesión se toma como un diseño inapropiado y difícil de entender. Además las clases con baja cohesión suelen contener más errores por lo que son menos tratadas ya sea por su nivel difícil de comprensión. "La métrica de cohesión de C&K se calcula tomando cada par de métodos en una clase. Si acceden a distintas variables de instancias, se aumenta P en uno. Si comparte por lo menos un valor, se aumenta Q en uno" [8].

$$LOCM = \begin{cases} P - Q, & si\ P \geq Q \\ 0, & caso\ contrario \end{cases}$$

Un valor alto en $LCOM$ indica que la clase estaría intentando cumplir con más de un objetivo, es decir posee más de una función. Si aplicamos POA, el nivel de cohesión decae, dado la orientación a aspectos ayuda a extraer los procedimientos y variables que se utilizan para realizar otras funciones (registro de operaciones, sincronización, etc.) que no son naturales para las clases que las implementan.

SISTEMA ORIENTADO A ASPECTOS

Un sistema orientado a aspectos debe cumplir una serie de propiedades [11]:

- Cada aspecto debe ser claramente identificable, auto-contenido y fácil de cambiar.
- Los aspectos no deben interferir entre ellos ni con la funcionalidad del sistema.

Uno de los problemas que se plantean en el desarrollo de software orientado a aspectos es la interferencia, interacción o conflicto entre aspectos.

Se dice que existe conflicto entre aspectos cuando un Joinpoint está asociado a diferentes pointcuts pertenecientes a distintos aspectos, esto es, que diferentes aspectos piden activarse en el mismo momento. Esta situación puede no ser un problema si los aspectos son independientes, es decir, que el comportamiento representado en cada aspect es independiente del resto, por lo que el comportamiento del sistema no se ve afectado.

LINEA DE INVESTIGACION Y PERFIL ORIENTADO A ASPECTOS

Haciendo énfasis en las líneas de investigación y desarrollo orientados a la Programación Orientada a Aspectos, se sustenta que "en la actualidad no existe un estándar para el soporte de aspectos" [12]. Por otro lado, "la OMG tiene estándares que soportan y permiten la definición de estereotipos a través de los Perfiles. Muchas herramientas soportan este estándar, y a través de estas herramientas es posible definir aspectos utilizando estereotipos para lograr modelos Orientado a Aspectos (OA)" [13].

De esta manera es posible la construcción de herramientas que genere código OA totalmente independiente de las herramientas utilizadas para el modelado que sigan con las especificaciones de la OMG. "*Un aspecto (aspect) es la unidad central en un lenguaje OA, de la misma manera que la clase es la unidad central en Programación Orientada a Objetos (POO)*" [14]. Por otro lado, los estándares de la OMG brindan la posibilidad de definir Perfiles los cuales hacen posible la construcción de diseños OA, ya que en la actualidad no existe un estándar en cuanto a estos diseños. A través de estos perfiles, es posible definir estereotipos para lograr el soporte de aspectos. La semántica que tendrán estos estereotipos parte de la correspondencia con el código OA, utilizando reglas OCL.

Las reglas OBJECT CONSTRAINT LANGUAGE (OCL), "*es un lenguaje de expresión que permite describir operaciones y restricciones sobre modelos Orientados a Objetos (OO)*

y otros artefactos de modelado" [15]. "OCL es un lenguaje tipado. Los tipos pueden ser cualquier clase de clasificadores dentro de UML. Cabe destacar que no es un lenguaje de programación, por lo que no es posible escribir lógica de programas o flujos de control, el mismo OCL es utilizado para definir la semántica de UML, especificando reglas bien formadas sobre el metamodelo. Las reglas bien formadas son definidas como invariantes sobre las metaclases en la sintaxis abstracta. También es usado para definir operaciones adicionales, cuando las reglas bien formadas las precisan" [16]. En las expresiones OCL, la palabra clave context introduce el contexto para una expresión: es decir en que clase se puede utilizar, cuál será la operación que se le va asignar o la operación que va a realizar dentro del sistema, que restricciones va a tener, como será el comportamiento en su entorno, cual es su precondición y su Postcondición.

"En la práctica hay muchas reglas sobre una clase en un modelo, las cuales no pueden ser especificadas gráficamente. Llamamos a esas reglas restricciones. Una restricción es una expresión que da información adicional al modelo visual o artefacto. Las restricciones se aplican a elementos de modelos o de sistemas Orientado a Objetos (OO) y siempre restringen valores de esos elementos. Las restricciones deben ser verdaderas en un momento dado para que el modelo o el artefacto sean válidos" [17].

Como se menciono al inicio de este artículo, cuando se desarrolla una aplicación se integran diversas tareas las cuales se deben realizar, y dentro de estas tareas se asocian los requerimientos funcionales del sistema los cuales hacen se pueden denominar tareas principales del sistema, por otro lado tenemos aquellas tareas que no están detalladas en el análisis funcional del sistema las cuales se consideran como servicios comunes. Cuando se habla de servicios comunes se hace referencia a cuando existe la posibilidad o la necesidad de realizar auditoria en el logging del sistema, es decir el acceso a la base de datos, los permisos o privilegios que posee cierto usuario al tener acceso a determinada información y todo lo relacionado con la seguridad del sistema para así saber que tan seguro es. Cada una de estas tareas es considerada una incumbencia, es decir que al momento de realizar cualquier acción o tarea en el código solo debe responder el método al cual pertenece esta tarea o la acción que se va a realizar. La separación de incumbencia es de suma importancia en el desarrollo de software ya que de esta manera el código es más entendible y reutilizable.

Las incumbencias son los diferentes temas, asuntos o aspectos de los que es necesario ocuparse para resolver un problema determinado. Cuando separamos las incumbencias se reduce la complejidad y son más entendibles al momento de interactuar con ellas.

El término separación de incumbencias fue introducido en la década de 1970, por Edsger W. Dijkstra significa, simplemente, que "*la resolución de un problema dado involucra varios aspectos o incumbencias, los que deben ser identificadas y analizados en forma independiente*" [17].

"La Programación Adaptativa fue el concepto predecesor de la Programación Orientada a Aspectos. Las ideas originales de la Programación Adaptativa fueron introducidas a principio de la década de 1990 por el grupo Demeter" [18], siendo Karl Lieberherr uno de sus ideólogos. Como lo definimos al inicio de este estudio del arte los conceptos de POA fueron introducidos en 1997 por GregorKiezales y su grupo [19].

Cuando se empieza a implementar la Programación Orientada a Aspectos, en los lenguajes tradicionales ya conocidos se implementan todas aquellas funciones principales tales como objetos, métodos o procedimientos. Los lenguajes orientados a aspectos definen una nueva unidad de programación de software para encapsular las funcionalidades diseminadas por todo el código (los aspectos). Sin embargo, componentes y aspectos deben interactuar. En la ejecución del programa, finalmente, los aspectos deberán estar insertados dentro de los componentes. Para ello será necesario definir claramente cómo será ésta estrategia de inserción. En la terminología de POA, este proceso se denomina entretejido, ya que puede pensarse que aspectos y componentes deben entretejerse para formar finalmente un código ejecutable.

Para que pueda existir el entretejido entre aspectos y componentes, debe ser necesario definir o declarar ciertos puntos de enlace, los llamados puntos de enlace son una clase especial de interfaz entre los aspectos y los módulos del lenguaje de componentes. El encargado de realizar la inserción de los aspectos en los puntos de

enlace es conocido como tejedor. El tejedor se encarga de tejer o entremezclar los diferentes mecanismos de abstracción y composición que aparecen en los lenguajes de aspectos y componentes en los puntos de enlace. Para tener un programa orientado a aspectos necesitamos definir los siguientes elementos [20]:

- "Un lenguaje para definir la funcionalidad básica. Este lenguaje se conoce como lenguaje base. Suele ser un lenguaje de propósito general, tal como C++ o Java. En general, se podríanutilizarcualquierlenguaje.
- Uno o varios lenguajes de aspectos. El lenguaje de aspectos define la forma de los aspectos.
- Un tejedor de aspectos. El tejedor se encargará de combinar los lenguajes. El proceso de mezcla puede hacerse en el momento de la compilación, o puede ser retrasado para hacerse en tiempo de ejecución".

En los lenguajes tradicionales, es suficiente tener un compilador o intérprete el cual se encarga de generar el código entendible por la máquina. En los lenguajes orientados a aspectos, además del compilador, es necesario tener un tejedor, el cual se encarga de combinar el código de los componentes con el código de los aspectos.

Una manera de tejer los componentes y los aspectos en el tiempo de ejecución es de forma estática, donde el entretejido producido consiste en la modificación del código fuente de los componentes que lo conforman. Al utilizar este tipo de entretejido se tienes como ventaja el evitar que el nivel de abstracción en el cual se introduce con la ayuda de la Programación Orientada a Aspectos se derive una degradación de la representación o diseño de la aplicación. Por otro lado se tiene el entretejido dinámico el cual requiere que los aspectos existan y que si estos existen deben están en el tiempo en que se está compilando como en el tiempo de ejecución, el tejedor dinámico permite durante la ejecución añadir, poder adaptar y remover aspectos, además pueden utilizarse mecanismos de herencia en forma dinámica para agregar el código de los aspectos a los componentes. Las desventajas radican en afectar directamente el diseño de la aplicación y en la complejidad de la depuración de errores, ya que varios de éstos podrían ser detectados únicamente en tiempo de ejecución. Anteriormente, la orientación a aspectos se centraba principalmente en desarrollo, pero con el avance y la necesidad de avanzar surgen trabajos para llevar la separación de incumbencias a nivel de diseño.

Varios trabajos [21] [22] [23] [24] proponen utilizar UML (UnifiedModelingLanguage) como lenguaje de modelado, ampliando su semántica con los mecanismos que el propio lenguaje unificado tiene para tales efectos y consiguiendo así representar la funcionalidad básica separada de los otros aspectos.

Al desarrollar un sistema basado en aspectos se debe tener claro que es lo que debe contener y que se va a compartir entre el lenguaje de aspectos, dentro del lenguaje de componentes se debe mirar cual es la forma más adecuada para poder implementar el funcionamiento principal y a su vez inspeccionar que si el programa escrito en ese lenguaje no puedan interferir con los aspectos es decir, manejar la separación de incumbencias. Los lenguajes de aspectos tienen que proveer los medios para implementar los aspectos deseados ya sea de manera intuitiva, natural y concisa.

El desarrollo de una aplicación basada en aspectos requiere de tres pasos [25]:

- Descomposición de aspectos y componentes: Descomponer los requerimientos para distinguir aquellos que son componentes de los que son aspectos.
- Implementación de las incumbencias: Implementar cada incumbencia por separado (aspectos y componentes).
- Recomposición: Definir las reglas que permitan combinar los aspectos con los componentes.

La actual tendencia es detallar la primera etapa de descomposición de componentes y aspectos en el momento de diseño, y no posponerlo a la etapa de desarrollo. La segunda etapa, de implementación, consiste en programar los componentes y los aspectos con sus respectivos lenguajes.

Dentro de los lenguajes orientados a aspectos la idea central de la POA es permitir el desarrollo de un programa en el cual se pueda describir cada concepto o incumbencia que tenga el programa por separado. El soporte para este paradigma se logra mediante la creación de nuevas clases de lenguajes llamados Lenguaje Orientado a Aspectos LOA.

Los LOA son aquellos lenguajes que permiten separar la definición de la funcionalidad principal de la definición de los diferentes aspectos.

Los LOA deben satisfacer varias propiedades deseables, entre las que se pueden mencionar [26]:

- Cada aspecto debe ser claramente identificable.
- Cada aspecto debe auto contenerse.
- Los aspectos deben ser fácilmente modificables.
- Los aspectos no deben interferir entre ellos.
- Los aspectos no deben interferir con los mecanismos usados para definir y evolucionar la funcionalidad principal, como la herencia.

LENGUAJES ORIENTADOS A ASPECTOS

Se distinguen dos enfoques diferentes en el diseño de los lenguajes orientados a aspectos: *los lenguajes orientados a aspectos de dominio específico y los lenguajes orientados a aspectos de propósito general.*

Los LOA de dominio específico son diseñados para el soporte de algún tipo de aspecto en particular, es decir, la manera en cómo se distribuyen y cuál es su concurrencia, este lenguaje de dominio específico posee un nivel de abstracción superior al de un lenguaje base. Muchas veces este tipo de lenguaje necesita restricciones en el lenguaje base, para así poder garantizar todas aquellas incumbencias que suelen ser tratadas en los aspectos y las cuales no pueden ser programadas en los componentes, evitando todas aquellas inconsistencias o funcionamiento no deseados e inesperados.

Los LOA de propósitos generales son diseñados para soportar cualquier tipo de aspectos, en este tipo de lenguaje no se tienen ningunas restricciones en el lenguaje base, ya que posee el mismo nivel de abstracción que el lenguaje base, *los LOA de propósito general* tienen la clara ventaja de ser, tal como su nombre indica, generales, ya que pueden ser utilizados para poder desarrollar con ellos cualquier tipo de aspecto.

Igual que los LOA de dominio especifico estos poseen una desventaja los cuales no garantizan la separación de las funcionalidades, y no se podrá realizar ninguna restricción respecto a dicho componente. Mientras que *los LOA de dominio específico* fuerzan a programar las tareas de aspectos dentro de éstos, ya que en el lenguaje base se restringe el uso de las instrucciones relacionadas con las funcionalidades de aspectos.

PROGRAMAS CON LENGUAJE DE DOMINIO ESPECÍFICO Y GENERAL

En los lenguajes de dominio específico se encuentra el lenguaje COOL, su principal objetivo es manejar los aspectos cuando se llega a la sincronización entre los llamados hilos concurrentes en el sistema. El lenguaje base utilizado por COOL es java en versión modificada es decir donde se eliminan los métodos tales wait este método consiste en paralizar o detener el programa en el tiempo que se ejecuta, y los métodos notify y notifyAll estos son los que permiten al programa enviar mensajes o notificaciones al usuario, todos estos tres métodos están incluidos en la palabra reservada synchronized para así evitar los inconsistencias que se puedan presentar al momento de sincronizar los hilos en el aspecto y en los componentes es decir las clases.

AspectJ es un LOA de propósito general [27], "que extiende Java con una nueva clase de módulos que implementan los aspectos. Los aspectos cortan las clases, las interfaces y a otros aspectos". 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. Cuando se habla de puntos de corte, son todas aquellas capturas de evento durante la ejecución del programa, en estos eventos es donde se hacen las llamadas a los métodos. Los cortes no definen acciones, sino que describen eventos. *"En forma similar a AspectC++, en AspectJ se definen cortes (pointcuts), avisos y puntos de enlace"* [28].

La primera decisión que hay que tomar al implementar un sistema orientado a aspectos es relativa a las dos formas de entrelazar el lenguaje componente con el lenguaje de aspectos. Dentro de las formas esta el entrelazado o tejido estático y el entrelazado o tejido dinámico.

El entrelazado estático implica que se modifique el código fuente que se escribió en el lenguaje base, para poder insertar sentencias en los puntos de enlace, esto permite que el código de los aspectos se implemente en

el código fuente, el ejemplo más claro de este tipo de tejedor de aspectos es el AspectJ.

En el entrelazado dinámico deben existir los aspectos y deben estar presentes durante el tiempo que se compila, y en el tiempo en que se ejecuta. El tejedor dinámico puede añadir, remover y adaptar aspectos cuando se estén ejecutando. Un ejemplo de este tipo de tejedor es el AOP/ST este utiliza la herencia para así poder añadir el código especifico que requieran las clases dependiendo de donde hereden, y así evitar que se modifique el código fuente de las clases principales al momento de entrelazar los aspectos.

"Otro ejemplo más completo sería el Junction Point AspectLanguage (JPAL), basado en los conceptos de protocolos de meta objetos (metaobjectprotocols MOP) y meta-programación. En JPAL existe una entidad llamada Administrador de Programas de Aspectos el cual puede registrar un nuevo aspecto de una aplicación y puede llamar a métodos de aspectos registrados" [29]. Este es implementado como una librería dinámica la cual permite almacenar los aspectos y puede añadir, modificar incluso quitar aspectos, y manda los mensajes a dichos aspectos en los cuales se pudo generar la acción.

El tejido estático evita que el nivel de abstracción introducido por la POA derive en un impacto negativo en la eficiencia de la aplicación, ya que todo el trabajo se realiza en tiempo de compilación, y no existe sobrecarga en ejecución. El tejido estático no permite que durante la ejecución los aspectos puedan ser modificados, se pueda agregar ni mucho menos removerlos ya que estos aspectos quedan fijos, no son modificables. La ventaja de este tejedor es que es seguro ya que cuando se compila no surgen errores los cuales puedan afectar el sistema durante la ejecución, a su vez el tejedor dinámico es más recursivo ya que consume pocos recursos.

Por otro lado está el tejedor dinámico el cual implica que el proceso de composición se realice en tiempo de ejecución para así poder disminuir la eficiencia de la aplicación. Cuando se posterga dicha composición se obtiene mayor flexibilidad y libertad para el programador, pues cuenta con la posibilidad de poder modificar un aspecto según la información que se produzca en el tiempo de ejecución, puede además introducir o remover el o los aspectos dinámicamente.

La característica dinámica de los aspectos pone en riesgo la seguridad de la aplicación, pues su puede remover de forma dinámica el comportamiento de un aspecto y que más adelante se requiera para la ejecución.

APORTES RELACIONADOS A CONFLICTOS ENTRE ASPECTOS EN ETAPA DEL DESARROLLO DEL SOFTWARE

En la siguiente tabla se indican brevemente los trabajos y aportes relacionados específicamente con la problemática de detección y resolución de conflictos entre aspectos.

Tabla 2. Algunos aportes de los conflictos entre aspectos [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41].

Propuesta	Detección	Resolución
[30] [31]	Análisis de conflictos estáticos.	Soporte lingüístico para resolución de conflictos.
[32]	Detección y análisis de las interferencias causadas en AspectJ.	
[33]		Mejora el modelo de precedencia de AspectJ.
[34]	Provee análisis de las interferencias aspecto-aspecto para AspectJ.	
[35]	Analiza las interacciones entre aspectos. Utiliza la técnica ProgrammeSlicing.	
[36]	Sigue el esquema propuesto por [31]. Se limita a una aproximación estática de la interacción de aspectos, solo se detectan las interacciones que no ocurren en tiempo de ejecución.	Dos formas de resolver una interacción (40) elegir de las interacciones el aspecto que se va a aplicar en la ejecución (41) ordenar y anidar los aspectos para la ejecución.
[37]	Análisis de interacciones entre aspectos. Utiliza Filtros de composición.	
[38]	Exploración inicial basada en lógica donde los hechos y reglas son definidos para la detección de interacciones en Réflex.	
[39]		Enfoque declarativo basado en restricciones para especificar la composición de

		aspectos ante un conflicto. Las restricciones pueden ser de orden o control, y pueden aplicarse en forma independiente y no combinarse.

TRATAMIENTO DE CONFLICTOS EN LA INGENIERÍA DE REQUERIMIENTOS

La Ingeniería de Requerimientos Orientada a Aspectos (AORE) [39] intenta proveer soporte para la identificación y separación de los propiedades funcionales y no funcionales.

"En esta fase del desarrollo los requerimientos transversales se denominan aspectos tempranos (earlyaspects)" [42]. "Otro objetivo de la AORE es proveer mejores medios para la identificación y manejo de conflictos que surjan entre los aspectos tempranos, lo que se ha denominado como conflictos tempranos (earlyconflicts)" [43].

En la Tabla siguiente se sintetizan los trabajos que se refieren a los aspectos tempranos y el tratamiento de las situaciones conflictivas que se generan a partir de ellos.

Tabla 3: Los trabajos tempranos realizados par los conflictos entre aspectos [44] [45] [46] [47] [48] [49].

Propuesta	**Detección**	**Resolución**
Requerimientos Orientados a Aspectos [44]	Manual	Prioridad
Ingeniería de Requerimientos Orientado a Aspectos [45]	Manual	Prioridad
Especificación y Separación de Concerns desde los requerimientos al Diseño [46]	Framework/Manual	Prioridad
PROBE [47]	Framework/Manual	Orden de preferencia y debilidades/Cambio de Requerimientos. Aclaración: se resuelve antes de la etapa de implementación.
Detección de conflictos entre CrosscuttingConcerns, basado en el Modelado [48]	Herramienta/Manual	No propone clasificación alguna. Aclaración: se resuelve en la etapa de implantación.
Aspects Extractor [49]	Automática	No propone clasificación alguna

OTROS AUTORES

Por otro se citan diversos autores que se apropian de los conflictos entre aspectos:

Douence proponen una solución basada en "*un framework genérico para POA, que se caracteriza por un lenguaje de cortes cruzados muy expresivo, análisis de conflictos estáticos y un soporte lingüístico para la resolución de conflictos*".

LogicAJ [51] provee análisis de las interferencias aspecto-aspecto para AspectJ que incluye capacidades para:

Identificar una clase bien definida de interferencias

Determinar el orden de ejecución libre de interferencia

Determinar el algoritmo de tejido más conveniente para un conjunto de aspectos dado

Fig. 1. Análisis de interferencias de aspecto a aspectos

Astor [52] *"es un prototipo que propone una serie de mecanismos y estrategias para mejorar el tratamiento de conflictos en AspectJ*".

"La detección de conflictos actúa por una clasificación de los mismos por niveles de semejanza y la resolución se efectúa siguiendo las directrices de una taxonomía que proporciona seis categorías de resolución" [53].
La implementación del prototipo está basada en el pre-procesamiento de código AspectJ, siendo además éste el único requisito para su uso.

ProgrammeSlicing es una técnica que apunta a la extracción de elementos de programa relacionados a una computación en particular. "*Este enfoque es propuesto para analizar las interacciones entre aspectos, ya que puede reducir las partes de código que se necesitan analizar para entender los efectos de cada aspecto*". [54]

Los Filtros de Composición se utilizan para analizar interacciones en [55]. En este trabajo se detecta cuando un aspecto precede la ejecución de otro aspecto, y se chequea que una propiedad especificada de traza sea realizada por un aspecto.

En [56] se presenta una exploración inicial basada en programación lógica donde los hechos y reglas son definidos para la detección de interacciones en Reflex.

Un modelo formal para la detección de conflictos directos y estimación del impacto, en la etapa diseño del desarrollo es presentado en [57]. Para identificar un conflicto se detalla la información generada a partir del diagrama de clases que se genera en UML, el cual por medio de sus extensiones incluye aspectos y clases. La información detallada tales como pointcuts, joinpoints, Advice es mapeada en tablas que permiten que un algoritmo identifique todas las posibles relaciones entre las clases y aspectos. Por medio del tipo de relación se obtiene una predicción del posible impacto del conflicto y su recomendación para solucionarlo.

El propósito de los diversos autores es identificar las interacciones o relaciones entre aspectos en la fase de modelado, para así proporcionar el método formal el cual permita detectar los conflictos.

CONCLUSIONES

Con el anterior estudio del arte se llega a la conclusión que la Programación Orientada a Aspectos es un nuevo paradigma de programación el cual maneja metodologías que ayudan en el momento en que se empieza desarrollar sistemas software, es decir permite el mantenimiento y seguimiento de este. Las metodologías brindadas para ser utilizadas deben ser evaluadas, clasificadas para saber cual se puede utilizar o cual es la que más se adapta a la necesidad del problema. Las metodologías brindan las herramientas necesarias para satisfacer la necesidad requerida, una de las herramientas permiten medir de manera más objetiva en el diseño realizado por medio de distintas técnicas, específicamente en los sistemas Orientados a Aspectos en este caso las métricas que son las que rigen este paradigma de programación. Con las métricas se lleva a profundizar aquellas métricas que son aplicables al diseño Orientado a Aspectos y los posibles efectos que esta genera, además se busca el desarrollo de nuevas métricas que ayuden en la calidad del desarrollo de sistemas Orientado a Aspectos.

Durante el estudio realizado se dijo que las métricas ayudan a mejorar el desarrollo de los sistemas basados en la programación orientada a aspectos. En el anterior estudio se presentaron aquellas métricas definidas por autores expertos en el tema y se llego a la conclusión que muchas de ellas se encuentran en estudio y que muchas de ellas ayudan a manejar los conflictos presentados entre aspectos.

Dentro de los conflictos entre aspectos la programación orientada a aspectos juega un papel importante para la resolución de conflictos pero si no se aplica de manera cuidadosa puede generar un conflicto mayor.

REFERENCIAS

[1] GregorKickzales, John Lamping, AnuragMendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin. "Aspect-Oriented Programming", European Conference on Object Oriented Programming (ECOOP), 1997.

[2] Universidad del Bío-Bío. Profesor Guía. Facultad de Ciencias Empresariales. Departamento de Sistemas de Información.

[3] METRICAS APLICADAS A LA PROGRAMACION ORIENTADA A ASPECTOS (Worskshop de Investigadores en Ciencias de la Computación WICC-2006) Lorena S. Baigorria, Germán Montejano Departamento de Informática, Universidad Nacional de San Luis San Luis- CP 5700- Argentina

[4] Junichi Suzuki, Yoshikazu Yamamoto. Extending UML with Aspect: Aspect Support in the Design Phase. 3er Aspect-OrientedProgramming (AOP) Workshop at ECOOP'99.

[5] Mira Mezini (Programación orientada a aspectos y el razonamiento modular).

[6] Cristina VideiraLopes (Programación orientada a aspectos).

[7] Karl Lieberherr (Programación orientada a aspectos con los métodos adaptativos), Stefan Hanenberg (¿Cuál es el impacto de la programación orientada a aspectos sobre el mantenimiento de software? Un estudio empírico con AspectJ y Java (Tesis de Licenciatura en Ciencias de la Computación, 2010).

[8] Shyam R. Chidamber, Chris F. Kemerer. A Metrics suite for Object Oriented design. M.I.T. SloanSchool of Management E53-315. 1993

[9] Daniela Glasberg, Khaled El Emam, WalcelioMelo, NazimMadhavji: Validating Object-Oriented Design Metrics on a Commercial Java Application. 2000.

[10] Houari A. Sahraoui, Robert Godin, ThierryMiceli: Can Metrics Help Bridging the Gap Between the Improvement of OO Design Quality and Its Automation.

[11] Administración de conflictos entre Aspectos en AspectJ. Sandra Casas, Claudia Marcos, VerónicoVanoli, HectorReinaga, Luis Sierpe, jane Pryor y Claudio Saldivia. In Proceedings of the Six Argentine Symposium on Software Engineering (ASSE´2005) ISSN 1666 1087, 34 JAIIO (JornadasArgentinas de Informática e InvestigaciónOperativa) ISSN 1666, pp 111-125. Rosario Agosto 2005.

[12] OMG Unified Modeling Language specification http://www.omg.org

[13] Jingjun Zhang; Yuejuan Chen; Guangyuan Liu; Modeling Aspect-Oriented Programming with UMLProfile Education Technology and Computer Science, 2009. ETCS '09. First International Workshop on

[14] N. Debnath L. Baigorria, D. Riesco, G. Montejano "Metrics Applied to Aspect Oriented Design Using UML Profiles" IEEE symposium on computers and communications 2008 (ISCC 2008) Press Marruecos.

[15] Jos Warner; AnnekeKleppe; "The Object Constraint Language" Precise Modeling with UML; Eddison Wesley Publishing 1999.

[16] OCL2.0 Specification, http://www.omg.org, Julio del 2008.

[17] On the role of scientific thought. Prof. Dr. Edsger W Dijkstra. Burroughs Research Fellow
30th August 1974
http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html

[18] Adaptive Object-Oriented Software - The Demeter Meted, Karl Lieberherr, College of Computer Science, Northeastern University Boston, 1996

[19] Aspect-Oriented Programming
GregorKiczales, John Lamping, AnuragMendhekar, Chris Maeda, Cristina
Videira Lopes, Jean-Marc Loingtier, John Irwin

[20] Visión general de la Programación Orientada a Aspectos
Antonia Mª Reina Quintero
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Sevilla
Diciembre, 2000

[21] Extending UML with Aspects: Aspect Support in the Design Phase
Junichi Suzuki, Yoshikazu Yamamoto
3rd Aspect-Oriented Programming (AOP) Workshop at ECOOP'99

[22] From AOP to UML: Towards an Aspect-Oriented Architectural Modeling Approach
Mohamed M. Kandé, JörgKienzle and Alfred Strohmeier
Software Engineering Laboratory
Swiss Federal Institute of Technology Lausanne
CH - 1015 Lausanne EPFL
Switzerland

[23] A UML Profile for Aspect Oriented Modeling
Omar Aldawud, TzillaElrad, AtefBader
OOPSLA 2001 workshop on Aspect Oriented Programming

[24] UML Profile Definition for Dealing with the Notification Aspect in Distributed Environments
José Conejero, Juan Hernández, Roberto Rodríguez
6th International Workshop on Aspect-Oriented Modeling
March 14, 2005, Chicago, Illinois, USA

[25] Want My AOP (Part 1, 2 and 3)
RamnivasLaddad
Java World, January 2002
http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html

[26] Programación Orientada a Aspectos, análisis del paradigma
Fernando Asteasuain, Bernardo Ezequiel Contreras
Tesis de Licenciatura
Departamento de Ciencias e Ingeniería de la Computación, UNIVERSIDAD NACIONAL DEL SUR, Argentina, Octubre de 2002

[27] AspectJ Project
http://www.eclipse.org/aspectj

[28] The AspectJ Programming Guide
http://www.eclipse.org/aspectj/doc/released/progguide/index.html

[29] L. Berger, "Junction Point Aspect: A Solution to Simplify Implementation of Aspect Languages and Dynamic Management of Aspect Programs", in Proceedings of ECOOP 2000, Junio de 2000, Francia.

[30] Duoence R., Fradet P., Südholt M. "Detection and Resolution of Aspect Interactions", Technical Report Nº4435, INRIA, ISSN 0249-6399, Francia, 2002.

[31] Duoence R., Fradet P., Südholt M. "A Framework for the Detection and Resolution of Aspect Interaction", In Proceeding of GPCE 2002, vol. 2487 of LNCS, USA, 2002, Springer Verlag, pp 173-188.

[32] Storzer M., Krinkle J. "Interference Analisys for AspectJ", FOAL: Foundations of Aspect-Oriented Languages, USA, 2003.

[33] Yu Y., Kienzle J. "Towards an Efficient Aspect Precedence Model", Proceeding of the DAW, pp 156-167, England, 2004.

[34] ROOTS. "LogicAJ – A Uniformly Generic and Interference-Aware Aspect Language". 2005. http://roots.iai.unibonn.de/researh/logicaj/.

[35] Monga M., Beltagui F., Blair L. "Investigating Feature Interactions by Exploiting Aspect Oriented Programming", Technical Report N comp-002-2003, Lancaster University, Inglaterra, 2003. http://www.com.lancs.ac.uk/computing/aop/Publications.php

[36] Tanter E., Noye J. "A Versatile Kernel for Multi-Language AOP", Proceeding of ACM SIGPLAN/SIGSOFT – Conference on GPCE. LNCS, Springer-Verlag, Estonia, 2005.

[37] Durr P., Staijen T., Bergmans L., Aksit M. "Reasoning about semantic conflicts between aspects". In K. Gybels, M. D'Hondt, I. Nagy and R. Douence, editors, 2nd European Interactive Workshop on Aspects in Software, 2005.

[38] Kessler B, Tanter E. "Analyzing Interactions of Structural Aspects". Workshop AID in 20th. ECOOP. France, 2006.

[39] Sampaio A., Loughran N., Rashid A., Rayson P. "Mining Aspects in Requirements", Workshop on Early Aspects, AOSD 2005.

[40] Homepage of AOSD: http://aosd.net/

[41] Homepage de AspectJ Xerox, PARC, USA http://aspectj.org/

[42] Homepage Early Aspects: Aspect Oriented Requirements Engineering and Architecture Design http://www.earlyaspects.net/

[43] Vanoli V., Marcos C. "Administración Temprana de Conflictos entre Aspectos". III WISBD. XII CACIC. Universidad Nacional de San Luis. Potrero de los Funes, San Luis. Octubre 2006. ISBN 950-609-050-5.

[44] Araújo J., Moreira A., Brito I., Rashid A. "Aspect-Oriented Requirements with UML". Workshop: Aspect-oriented Modeling with UML. Dresden, Germany. October 2002.

[45] Brito I. "Aspect-Oriented Requirements Engineering". UML. Lisbon, Portugal. October 2004.

[46] Kassab M., Constantinides C., Ormandjieva O. "Specifying and Separating Concerns from Requirements to Design: A Case Study". International Multi-Conference on Automation, Control, and Information Technology, Anaheim, California, USA: ACTA Press. pp. 18-27. 2005

[47] Katz S., Rashid A. "From Aspectual Requirements to Proof Obligations for Aspect-Oriented Systems". International Conference on RE, Japon, IEEE Computer Society Press. Pp 48-57, 2004.

[48] Tessier F., Badri M., Badri L. "A Model-Based Detection of Conflicts Between Crosscutting Concern: Towards a Formal Approach". International WAOSD. China, 2004.

[49] Haak B., Díaz M., Marcos C., Pryor J. "Aspects Extractor: Identificación de Aspectos en la Ingeniería de Requerimientos". IDEAS'06 9° Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software. La Plata, Argentina, 2006.

[50] Douence, R., Fradet, P., Südholt, M. "A Framework for the Detection and Resolution of Aspect Interactions" (2002). Proceedings of the ACM SIGPLAN SIGSOFT Conference on GPCE.

[51] ROOTS (2005) "LogicAJ – A Uniformly Generic and Interference-Aware Aspect Language"; http://roots.iai.uni-bonn.de/researh/logicaj/.

[52] Casas, S., Marcos, C., Vanoli, V., Reinaga, H., Saldivia, C., Prior, J. y Sierpe, L. (2005) "ASTOR: Un Prototipo para la Administración de Conflictos en AspectJ", XIII Encuentro Chileno de Computación, Jornadas Chilenas de Computación (JCC 05), Chile.

[53] Pryor, J., Marcos, C. (2003) "Solving Conflicts in Aspect-Oriented Applications". Proceedings of the Fourth ASSE. 32 JAIIO. Argentina.

[54] Monga, M., Beltagui, F., Blair, L. (2003) "Investigating Feature Interactions by Exploiting Aspect Oriented Programming", TR N comp-002-2.003, Lancaster University, Inglaterra. http://www.com.lancs.ac.uk/computing/aop/Publications.php

[55] Durr, P., Staijen, T., Bergmans, L., Aksit, M. (2005) "Reasoning about semantic conflicts between aspects". In K. Gybels, M. D'Hondt, I. Nagy, and R. Douence, editors, 2nd European Interactive Workshop on Aspects in Software.

[56] Kessler, B. and Tanter, E. (2006) "Analyzing Interactions of Structural Aspects". Workshop AID in 20th. European Conference on Object-Oriented Programming (ECOOP). France.

[57] Tessier, F., Badri, M., Badri, L. (2004) "A Model- Based Detection of Conflicts Between Crosscutting Concern: Towards a Formal Approach", International Workshop on Aspect – Oriented Software Development (WAOSD 04), China.