Modelo de Asociaciones: un enfoque para el tratamiento de conflictos entre aspectos

Sandra Casas[1], J. Baltasar García Pérez-Schofield[2], Claudia Marcos[3]

[1]Unidad Académica Río Gallegos, Universidad Nacional de la Patagonia
Lis. De la Torre 1060 – CP 9400 - Rio Gallegos - Argentina

[2]Universidad de Vigo – Departamento de Informática
Orense, España

[3]Instituto de Sistemas de Tandil, Universidad Nacional del Centro
Tandil, Argentina

lis@uarg.unpa.edu.ar, jbgarcia@uvigo.es, cmarcos@exa.unicem.ar

Resumen. La ocurrencia de ciertos conflictos entre aspectos puede producir comportamientos inciertos e indeseables en la ejecución del software, por lo que es necesario disponer de mecanismos específicos para controlar y administrar las situaciones conflictivas. La estructura de las herramientas de programación orientadas a aspectos puede influir en este sentido, flexibilizando o limitando la tarea. En este trabajo se propone el Modelo de Asociaciones como un enfoque alternativo que flexibiliza, amplia y automatiza la administración de conflictos entre aspectos.

Abstract. The conflicts among aspects can produce uncertain and undesirable behaviors in the execution of the software. Therefore, specific mechanisms to control and to handle the conflicting situations are necessary. The structure of the AOP tools can influence in this sense because of its flexibility or restrictions. In this work it is proposed the Model of Associations such as an alternative approach which is more flexible, wide and automatic way to solve the conflicts among aspects.

1. Introducción

La Programación Orientada a Aspectos (Kiczales et al., 1997) (POA) ha sido propuesta como una técnica para mejorar la Separación de Concerns (Dijkstra, 1976) (Hürsch & Lopez, 1995) (SoC). La idea central consiste en implementar los requerimientos transversales ("crosscutting concerns") en unidades separadas y aisladas denominadas aspectos. Una amplia variedad de herramientas, compiladores y frameworks, dan soporte al paradigma POA, siendo AspectJ (Kiczales et al., 2001) la herramienta POA más popular y difundida, no solo por su aplicación sino además su modelo de estructura ha sido replicado en otras herramientas.

Sin embargo, el modelo AspectJ plantea una estructura en el que la implementación de los requerimientos transversales se basa central y fuertemente en la entidad aspecto, como única unidad de implementación y declaración. Este factor impone fuertes restricciones y limitaciones para el manejo adecuado de conflictos entre aspectos. El aspecto debe: (i) encapsular la lógica o comportamiento del requerimiento transversal ("advices"); (ii) incluir la definición o declaración de la invocación aspectual ("pointcuts"); y (iii) establecer la resolución de conflictos ("declare precedence"). El cumplimiento de estas responsabilidades se debilita y resulta en consecuencia deficiente, ya que impacta negativamente en el desarrollo (diseño y codificación) y mantenimiento de las aplicaciones. El manejo de conflictos entre aspectos es sumamente acotado, no pudiendo dar un tratamiento adecuado en las siguientes situaciones: (i) detección automática de conflictos; (ii) proporcionar distintas posibilidades de resolución de conflictos (además del orden); (iii) proveer solución cuando un conjunto de aspectos provocan más de un conflicto y cada conflicto requiere distintas políticas de precedencias; (iv) proveer solución cuando el conflicto es planteado por distintos puntos de corte del mismo aspecto; (v) posibilitar que el orden de ejecución de los aspectos en conflicto dependa de una condición del sistema o del contexto; (vi) facilitar la determinación del orden de ejecución de los aspectos en conflicto, sin recurrir a la inspección de todos los aspectos (buscando declaraciones de precedencia); (vii) posibilitar la reutilización de los aspectos en conflicto que requieren distintas precedencias en diferentes aplicaciones.

Las problemáticas planteadas, conducen a la necesidad de establecer y definir modelos POA alternativos que superen estas dificultades y permitan manejar los conflictos entre aspectos de manera flexible, amplia y automática. Este trabajo presenta el Modelo de Asociaciones, cuyo objetivo es dar soporte a la SoC y mejorar el manejo de conflictos entre aspectos.

La estructura de este trabajo es la siguiente: en las Secciones 2 y 3 se presenta el Modelo de Asociaciones, en la Sección 4 se brindan algunos detalles de la implementación; en la Sección 5 se realiza una comparación entre el Modelo de AspectJ y el Modelo de Asociaciones; en la Sección 6 se resumen los trabajos relacionados y en la Sección 8 se presentan las conclusiones.

2 Modelo de Asociaciones

El Modelo de Asociaciones propone incorporar tres nuevas entidades: aspectos, asociaciones y reglas. Estas entidades permiten cumplir con todas responsabilidades necesarias para dar soporte a la SoC: (i) encapsular la lógica o comportamiento de los

requerimientos transversales en unidades separadas a la funcionalidad básica (aspectos); (ii) proporcionar mecanismos (sintácticos y semánticos) de extensión e invocación aspectual implícitos (asociaciones); (iii) establecer métodos y estrategias de resolución de conflictos amplios, flexibles y automáticos (reglas). A continuación se explican los conceptos de aspectos, asociación y regla, usando el lenguaje Java.

2.1. Aspectos

Un aspecto es una unidad de implementación que encapsula el comportamiento o lógica de un requerimiento transversal. Un aspecto se compone de la definición de un conjunto de propiedades (métodos y atributos). Un aspecto es una entidad de primera clase. En el Modelo de Asociaciones un aspecto solo incluye la extensión aspectual y no incluye la definición de la invocación aspectual. En este sentido, un aspecto internamente no contienen información alguna sobre el componente de funcionalidad básica que cortará transversalmente. En la Figura 1 se ha representado el aspecto *Logging*. Este aspecto se compone de los métodos *consoleLogOperation()* y *fileLogOperation()*. Como se puede observar, no existe ninguna declaración o definición que indique dónde y cuándo éste será aplicado. En principio esta característica, favorece la reutilización del aspecto en si mismo. Una vez compilado podría ser utilizado en distintas aplicaciones.

```
aspect Logging {
  public void consoleLogOperation(String idC, String IdO)
  { // enviar mensaje a consola  }
  public void fileLogperation(String idC, String IdO)
  { // enviar mensaje a fichero   }

}
```

Figura 1. Aspecto Logging.

Los aspectos al no incluir explícitamente los mecanismos de invocación aspectual admiten mayor nivel de reutilización. Sin embargo, no siempre podrá lograrse la reutilización total de todos los aspectos. Por ejemplo, este sería el caso de una aplicación que almacene y recupere toda la información que maneja en una base de datos relacional. Determinados detalles que son específicos de la base de datos relacional y la aplicación serán explicitados en los aspectos.

2.2. Asociaciones

Una asociación es una entidad que define una relación aspectual entre un miembro de clase y una propiedad de aspecto. Una asociación es una unidad que se declara de manera separada y aislada a las demás unidades de implementación (aspectos y clases). Una asociación es una unidad de declaración o definición, a diferencia de los aspectos y clases que son unidades de implementación.

```
1 association LogAccount
2 {
3   after public void Account.setBalance(float);
4   call public void Logging.consoleOperation(“Account”, “setBalance”);
5   priority = 10;
6 }
```

Figura 2. Asociación entre el aspecto Logging y la clase Account.

Por ejemplo, en la Figura 2 se define la asociación *LogAccount* que establece una relación aspectual entre la clase *Account* y el aspecto *Logging*. En particular se

establece que después que se ejecute el método *Account.setBalance(..)* se ejecutará el método *Logging.consoleOperation(..)*. En la línea 1 se define la palabra reservada association que indica el tipo de unidad y el identificador de la asociación; en la línea 3 se define la relación de aviso (*after*) y el miembro de clase sobre el que se realizará la extensión aspectual. La línea 4 se define la relación de corte (*call*) y la invocación aspectual implícita a una propiedad de aspecto. En la línea 5 se define la prioridad de la asociación. Esta propiedad solo afecta a la asociación, no al aspecto.

Una asociación es una relación *1-1*. Esto significa que sí un miembro de clase debe ser relacionado a otras propiedades de aspectos, deben definirse otras asociaciones. De manera similar, si una propiedad de aspecto corta transversalmente distintos miembros de clase, deben definirse distintas asociaciones. Los miembros de clase ("join-points") se definen por extensión, pero pueden ser definidos por comprensión (cuantificación), mediante el uso de comodines. En este caso, deben proveerse mecanismos adicionales automáticos que conviertan este tipo de relaciones *n-1* en relaciones *1-1*.

2.3 Reglas de Resolución de Conflictos

Un conflicto se produce cuando n asociaciones ($n \geq 2$) definen la misma relación de corte y de aviso sobre el mismo miembro de clase. Pero una asociación puede participar a lo sumo de un conflicto. Una propiedad de aspecto puede participar de distintos conflictos, al igual que un miembro de clase. Una regla de resolución de conflictos es una entidad que define una acción de resolución para m conflictos de un programa ($m \geq 1$).

$$R = \frac{K(k_1, k_2, \ldots, k_m) \quad // \text{antecedente}}{AR\ ((a_{11}, a_{12}, .., a_{1n}), .. (a_{m1}, a_{12}, .., a_{mn}))\ // \text{consecuente}}$$

Figura 3: Definición de Regla

Una regla de resolución se expresa claramente en dos partes (Figura 3). La parte del antecedente que identifica el conjunto de conflictos que se van a resolver. Y la parte del consecuente que especifica una acción precisa de resolución sobre las asociaciones en conflicto (no sobre los aspectos). Se distinguen dos tipos de reglas: reglas explícitas y reglas simbólicas. En ambos casos el propósito es el mismo, permitir aplicar una acción de resolución a m conflictos. La diferencia entre los distintos tipos de reglas radica en su especificación y en la implementación de las estrategias de detección y resolución de conflictos.

Las reglas explícitas requieren que el antecedente sea especificado de manera taxativa. Es decir, se debe indicar que asociaciones participan del conflicto. En el consecuente de la regla, estas asociaciones se especifican como parte de la acción de resolución (Figura 4).

$$RE = \frac{k(a_1, a_2, a_3)}{AR\ (a_1, a_2, a_3)}$$

Figura 4: Regla Explícita

Por ende, una regla explícita sirve para resolver un único conflicto. Utilizando esta estrategia debe especificarse por cada conflicto existente, una acción de resolución. En consecuencia el proceso de detección de conflictos debe realizarse previamente.

Las reglas simbólicas permiten aplicar una acción de resolución a subconjuntos de conflictos. Las reglas simbólicas pueden ser absolutamente generales o parcialmente generales. Cuanto más general sea el antecedente de la regla, más conflictos abarcará la acción de resolución definida en el consecuente. A medida que el antecedente sea más específico, menos conflictos serán afectados por el consecuente de la regla. La especificación de una regla simbólica no requiere conocer o definir el conflicto. El conflicto será conocido en el momento que se aplique la regla (Figura 5).

$$RS = \frac{(\text{existe el conflicto } x)}{AR\ (\text{ asociaciones de } x)}$$

Figura 5: Reglas Simbólica

Donde **x** puede referirse a todos los conflictos, o sólo a algunos de ellos. Por ejemplo: los conflictos sobre la clase *Account* o los conflictos sobre el método *Account.setBalance()*, o los conflictos en los que participa el aspecto *Logging*. En este caso, la regla simbólica es similar a las reglas de un sistema de producción, ya que el antecedente se plantea como una condición. El consecuente asumirá valores concretos de asociaciones luego de la detección de conflictos. Si la condición que debe satisfacer el conflicto definido en el antecedente de la regla simbólica no se cumple, simplemente la regla no se aplica a ninguna asociación. Dado que la especificación de reglas simbólicas implica que no se requiere conocer la existencia del conflicto en el momento de su definición, es necesario establecer algún criterio que permita aplicar las acciones de resolución a las asociaciones en conflicto. Este criterio es la prioridad de las asociaciones. Todas las asociaciones tienen una prioridad definida por el desarrollador, que será utilizada para la resolución de conflictos mediante reglas simbólicas. Por ejemplo en la Figura 6 se la reglas simbólicas *RS1*. En dicha regla la acción de resolución indica que las asociaciones de los conflictos deben ser ordenados de acuerdo a la prioridad de las mismas.

$$RS1 = \frac{(\text{existe conflicto } x)}{\text{Ordenar por prioridad (asociaciones de } x)}$$

Figura 6: Ejemplo de Regla Simbólica

La definición de prioridades en las asociaciones puede parecer un criterio arbitrario, sin embargo no lo es. A priori, el desarrollador sabe que ciertos aspectos por su naturaleza y comportamiento deben ser ejecutados primero o último, aunque desconozca con que otros aspectos está en conflicto. Por ejemplo, si una operación requiere la autentificación del usuario por cuestiones de seguridad, no tiene sentido que se realice ninguna otra operación antes de la autentificación. De la misma manera ocurre con los aspectos de persistencia y/o logging, por lo general, éstos deben ejecutarse siempre últimos. Esto significa que la definición de prioridades en las asociaciones esta relacionada a la naturaleza de los requerimientos transversales, que es conocida y comprendida por el desarrollador al momento de definir las asociaciones.

3. Taxonomía de Resolución

Las acciones de resolución conforman una taxonomía de resolución que establece diferentes categorías. Una categoría de resolución establece un método para resolver un conflicto. La definición de una taxonomía formada por un conjunto de categorías enmarcan a esta estrategia como amplia y flexible, superando el mecanismo convencional de orden. La taxonomía de resolución que se propone consiste en 5 categorías de acción de resolución de conflictos. En la Tabla 1 se ejemplifican y explican la aplicación de las categorías de la taxonomía para el conflicto formado por las asociaciones *a* y *b*.

Tabla 1. Acciones de las categorías de la taxonomía de resolución

Categoría	Ejemplo	Acción de Resolución
Orden	`Order(a, b);`	La propiedad de aspecto definida en la asociación a se ejecutará antes que la propiedad de aspecto b.
Orden Inverso	`iorder(a, b);`	La propiedad de aspecto definida en la asociación b se ejecutará antes que la propiedad de aspecto a.

Opcional	`if ( cond )`
	`  (a);`
	`else`
	`  (b);`
		Si cond es verdadera se ejecutará la propiedad de aspecto a, pero si cond es falsa se ejecutará La propiedad de aspecto b.

Exclusión	`excluded(a);`	Cuando se ejecuta el miembro de clase definido en la asociación a, la propiedad de aspecto definida en la asociación a no se ejecutará.
Nulidad	`anulled(a, b);`	Cuando se ejecute el miembro de clase definido en la asociación a y b, las propiedades de aspectos definidas en a y b no se ejecutarán.

Otra característica importante, es que las reglas explícitas permiten ampliar la Taxonomía (Tabla 1) a una Taxonomía de Combinación de categorías de resolución. En la Tabla 2 se exponen algunos ejemplos de las combinaciones posibles (*a*, *b* y *c* son asociaciones en conflicto).

Tabla 2. Taxonomía combinada.

Orden-Nulidad	Opcional-Orden	Opcional-Orden-Exclusión	Opcional-Orden-Exclusión-Nulidad

`order(a, b);`
`anulled (c);`
	`if (cond)`
	`  order (a, b, c);`
	`else`
	`  order (c, b, a);`
		`if (cond)`
		`  order (a, b, c);`
		`else`
		`  excluded (c, a);`
			`if (cond)`
			`  order (b, c);`
			`  excluded (a);`
			`else`
			`  anulled (c, a, b);`

El uso de prioridades en las acciones de resolución conduce a una especialización de la taxonomía original. En la Tabla 3 se definen las subcategorías que pueden ser aplicadas mediante reglas simbólicas. Las asociaciones *a*, *b* y *c* están en conflicto y tienen prioridades 5, 10 y 15 respectivamente.

Tabla 3. Categorías y sub-categorías de resolución para reglas simbólicas.

Categoría	Sub-categoría	Ejemplo
Orden	OBP: Ordenar las asociaciones en conflicto por prioridad.	`OBP (c, b, a);`
	IOBP: Ordenar las asociaciones en conflicto en forma inverso a la prioridad	`IOBP(a, b, c);`
	OF: Ordenar una asociación primero independientemente de	`OF (b, a, c);`

	su prioridad.	
	OL: Ordenar una asociación última independientemente de su prioridad.	`OL (a, c, b);`
Exclusión	ELP: Excluir la asociación en conflicto de menor prioridad.	`ELP (a);`
	EGP: Excluir la asociación en conflicto de mayor prioridad.	`EGP (c);`
	ETA: Excluir ésta asociación en conflicto.	`ETA (b);`
	EAA: Excluir todas las asociaciones en conflicto excepto ésta.	`EAA (a);`

La categoría de resolución de nulidad no requiere especialización ya que por la característica de la acción (eliminar del tejido todas las asociaciones del conflicto) la prioridad no tiene ningún uso o significado. La categoría de resolución opcional no es aplicable en las reglas simbólicas, debido a que se asume que la condición que forma parte de la acción, es específica a una operación de la funcionalidad básica y por ende a un conflicto.

Es importante recalcar que una regla simbólica particular no solo puede aplicarse a más de un conflicto, sino además que cada uno de estos conflictos pueden estar formado por cantidades de asociaciones diferentes. Estos dos factores se desconocen al momento de especificar la regla simbólica. La acción de resolución debe poder ser aplicada en todos los casos. Esta característica da mayor poder a las reglas simbólicas, pero requiere necesariamente de métodos de implementación adicionales más complejos.

En la Tabla 4 se comparan las reglas explícitas y las reglas simbólicas en cuanto a sus fortalezas y debilidades. Lo interesante aquí, es que las debilidades de las reglas simbólicas son superadas por las reglas explícitas y viceversa. De esta forma, las reglas explícitas y simbólicas constituyen métodos complementarios y no excluyentes que finalmente establecen una estrategia de resolución de conflictos entre aspectos sumamente flexible.

Tabla 4. Fortalezas y debilidades de reglas explícitas y reglas simbólicas

Regla	Fortaleza	Debilidad
Explícita	-Aplicación de categoría Opcional. -Aplicación de categorías de resolución combinadas. -Se sabe que conflicto se esta resolviendo en el momento de la especificación. -No requiere revisión posterior.	-Requiere especificación de una regla por conflicto, lo cual es más costoso ya que es manual.
Simbólica	-Definición de reglas generales o parcialmente generales. -Una regla puede resolver varios conflictos. -Utilización de criterio de prioridades.	-Restricciones en cuanto a las categorías de resolución aplicables. -Requiere revisión posterior.

Otras diferencias entre las reglas explícitas y reglas simbólicas se pueden establecer a partir del análisis de en que casos se debe usar cada una. Si ambos tipos de reglas se pueden aplicar indistintamente para resolver las mismas situaciones conflictivas, no tiene sentido utilizar distintos tipos de reglas. En la Tabla 5, se establecen estos criterios.

Tabla 5. Criterios para el uso de las reglas explícitas y simbólicas.

Situación	Regla Explícita	Regla Simbólica
Aplicar la categoría de resolución Opcional.	X	

Aplicar las categorías de resolución combinadas.	X	
Resolver un único conflicto	X	
Aplicar a subconjuntos de conflictos la misma categoría de resolución.		X
Los requerimientos admiten establecer prioridades en las asociaciones		X
Pocos conflictos.	X	
Muchos conflictos.		X

Independientemente de estas recomendaciones, en una aplicación se pueden aplicar ambos tipos de reglas. Como ya se indicó, las reglas son complementarias, posibilitando de esta forma un manejo de conflictos amplio, flexible y poderoso.

4. Implementación

MEDIATOR es un prototipo que da soporte al Modelo de Asociaciones. MEDIATOR extiende el lenguaje Java, para que se puedan definir asociaciones y ambos tipos de reglas. El procesamiento simbólico es realizado por un sistema experto de reglas embebido (Casas et al., 2006), que ha sido implementado en JESS (Friedman-Hill, 2003).

El tejido de los aspectos y clases, se realiza en dos partes. Primero, durante el proceso de compilación, se verifican y resuelven automáticamente posibles interferencias entre reglas (para los casos que existan dos sentencias de resolución para el mismo conflicto) y luego el tejedor de aspectos crea una "clase de enlace". Esta clase de enlace consta de un "método de enlace" por asociación (libre de conflicto) o sentencia de resolución (una sentencia de resolución se corresponde con un conflicto). El método de enlace consiste en una invocación directa a una propiedad de aspecto o un conjunto de sentencias que se corresponden con una categoría de resolución aplicada a un determinado conflicto. Por ejemplo en la Figura 7 el método de enlace *linking_method1()* invoca al aspecto *Logging.consoleOperation()*, mientras que el método *linking_method2()* invoca a dos aspectos diferentes en un determinado orden.

```
public void static linking_method1()
{
  Logging.consoleOperation(…);
}
```

```
public void static linking_method2(..) {
  Statistic.register();
  Logging.consoleOperation();
}
```

Figura 7: Ejemplos de métodos de enlace

Los métodos de enlace encapsulan las relaciones aspectuales y las resoluciones de conflictos, y los aspectos permanecen inalterables. El segundo paso, se realiza, durante la carga-ejecución de la aplicación. En el bytecode de las clases que están definidas en alguna asociación o regla se inserta una invocación (antes o después) al método de enlace correspondiente. La información básica para el proceso de tejido es el conjunto de las asociaciones y las sentencias de resolución producidas por las reglas de resolución, este proceso ha sido explicado en (Casas et al., 2007).

5 Modelo AspectJ vs. Modelo de Asociaciones

En la Tabla 6 se expone una comparativa entre el Modelo AspectJ y el Modelo de Asociaciones en cuanto al manejo de conflictos entre aspectos.

Tabla 6. Modelo Aspectj vs. Modelo de Asociaciones

Problemas	Modelo AspectJ	Modelo de Asociaciones
Detección de Conflicto	Manual	Automática
Categorías de Resolución	Solo orden (precedencia)	Orden – Orden Inverso – Opcional – Exclusión – Nulidad - Combinación de categorías.
Los conflictos son provocados por los mismos aspectos y cada conflicto requiere distinta política de orden.	No tiene solución. Deben re-codificarse los aspectos.	No debe re-codificarse ninguna unidad de implementación. Debe especificarse una regla explícita para cada conflicto.
El conflicto es planteado por distintos puntos de corte del mismo aspecto.	No tiene solución. La declaración de precedencia no tiene ámbito para puntos de corte del mismo aspecto. Debe re-codificarse el aspecto.	No debe re-codificarse ninguna unidad. Se puede aplicar cualquier categoría de resolución de la taxonomía mediante la definición de una regla explícita o simbólica.
El orden de ejecución depende de una condición el sistema o del contexto.	No tiene solución. Deben re-codificarse los aspectos.	Se debe especificar una regla explícita que aplique a categoría de resolución opcional.
Determinar el orden de ejecución de los aspectos.	Requiere esfuerzo y tiempo ya que debe inspeccionarse el código de todos los aspectos buscando las declaraciones de precedencia.	Solo deben analizarse las reglas.
Reutilización de los aspectos.	Reutilización Baja: 1) La declaración de los puntos de corte y tipo de aviso en el aspecto dificulta su reuso en otras aplicaciones, que se requiera aplicar a otros puntos de unión. 2) Si los aspectos en conflicto deben ejecutarse en distinto orden en otra aplicación, entonces no pueden ser reusados y deben modificarse el código.	Reutilización Alta: 1) La relación aspectual se define en las asociaciones, no en los aspectos. Sí esta cambia, se modifica la asociación, el aspecto permanece inalterable. 2) Si los aspectos en conflictos requieren distinta resolución en otra aplicación, solo debe modificarse las reglas, el aspecto permanece inalterable.

6. Trabajos Relacionados

(Douence et al., 2002) propone una solución que esta 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.

El modelo de precedencia de AspectJ (secuencial), es mejorado y optimizado en (Yu & Kienzle, 2004). Este trabajo es una propuesta de mejora para el mecanismo de resolución de conflictos específica para AspectJ. LogicAJ (ROOTS, 2005) provee análisis de las interferencias aspecto-aspecto para AspectJ. Astor (Casas et al., 2005) es un prototipo que propone una serie de mecanismos y estrategias para mejorar el tratamiento de conflictos en Aspecto, incluye detección y resolución de conflictos.

La técnica Programme Slicing es propuesta por (Monga et al., 2003) 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. Los Filtros de

Composición se utilizan para analizar interacciones en (Durr et al., 2005). 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. Ambos trabajos solo dan cobertura a la detección de conflictos.

Reflex (Tanter & Noye, 2005), una herramienta que facilita la implementación y composición de diferentes lenguajes orientados a aspectos. Según los autores, lo atrayente del modelo es que provee un alto nivel de abstracción para implementar los nuevos lenguajes de aspectos y soportar la detección y resolución de conflictos. Reflex consiste básicamente en un kernel cuya arquitectura dispone de 3 capas: (1) una capa de transformaciones encargada del tejido básico con soporte para la modificación estructural y de comportamiento de programas bases; (2) una capa de composición para la detección y resolución de interacciones y (3) una capa de lenguaje, para la definición modular del lenguaje de aspectos. La detección de interacciones sigue el esquema propuesto por (Douence, 2002) y se limita a una aproximación estática de la interacción de aspectos, solo se detectan las interacciones que no ocurren en tiempo de ejecución. Las dos formas de resolver una interacción son: (1) elegir de las interacciones el aspecto que se va a aplicar en la ejecución (2) ordenar y anidar los aspectos para la ejecución.

Un método para validar formalmente el orden de precedencia entre aspectos que comparten un mismo punto de unión, es presentado en (Pawlak et al., 2005). Este trabajo introduce un lenguaje simple, CompAr, en el cual el usuario expresa el efecto de los avisos que es importante en la interacción entre aspectos y las propiedades que deben ser verdaderas luego de la ejecución de un aviso. El compilador CompAr chequea que dado un orden de avisos no invalide una propiedad de un aviso de tipo “around”.

El uso de reglas como estrategia o mecanismo para manejar conflictos ha sido propuesto recientemente en varios trabajos. (Kessler & Tanter, 2006) presenta una exploración inicial basada en lógica donde los hechos y reglas son definidos para la detección de interacciones en Reflex. En (Nagy et al., 2006) se propone un enfoque declarativo basado en restricciones para especificar la composición de aspectos ante un punto de unión compartido (“shared join-points”). Las restricciones pueden ser de orden o control, y pueden aplicarse en forma independiente y no combinarse. La implementación de esta modelo requiere a extensión del lenguaje POA en varios aspectos: constructores de punto de unión, constructores de avisos, sentencias de declaración, etc. En consecuencia, las restricciones del lenguaje POA impactan sobre el modelo.

El Modelo de Interacciones (Charfi et al., 2006) presenta similitudes con el Modelo de Asociaciones. Los autores proponen representar los aspectos como componentes y las relaciones aspectuales en unidades denominadas “Interaction”. Las Interactions son una fusión de asociaciones y reglas explícitas. Se proporciona un lenguaje de especificación de Interacciones (ILS) que consiste en conjunto de operadores que permiten definir reglas de tejido (“merging”). Los operadores permiten aplicar distintas resoluciones a los conflictos: ejecución condicional, exclusión mutua y orden y la composición secuencial o concurrente. Aquí es importante que todas las relaciones aspectuales sobre un mismo punto de unión son mezcladas en una única interacción (unifica avisos after y before). Las interacciones se definen en forma individual, luego un proceso de mezcla automático genera la interacción final.

7. Conclusiones y Trabajo Futuro

El modelo AspectJ presenta serias dificultades para el tratamiento de conflictos, ya que se plantean una serie de situaciones cuyas soluciones obligan al re-diseño y re-implementación del código; hacen más costoso el mantenimiento y reuso del software y en ciertos casos simplemente no tiene solución. La productividad y calidad del desarrollo de software se ven negativamente impactadas.

Este artículo propone el Modelo de Asociaciones, que plantea una forma diferente de definir las relaciones aspectuales. El modelo establece una clara separación entre unidades de declaración y unidades de implementación. Las clases y los aspectos se ubican como unidades de implementación, mientras que las entidades asociaciones y reglas se ubican como unidades de declaración. Esta forma de concebir una aplicación orientada a aspectos favorece el manejo de conflictos, promueve la reutilización del código y facilita el posterior mantenimiento.

Las reglas constituyen un poderoso mecanismo que posibilitan la aplicación de distintas acciones de resolución, pudiendo estas ser: simples, combinadas o subcategorías. Asimismo las reglas pueden ser explícitas o simbólicas. Las reglas explicitas permiten resolver un único conflicto y las reglas simbólicas permiten resolver subconjuntos de conflictos.

El Modelo de Asociaciones ha sido implementado en el entorno de programación MEDIATOR. MEDIATOR consiste en un conjunto de componentes, siendo uno de ellos el sistema experto de reglas embebido que realiza todo el procesamiento simbólico. MEDIATOR permite utilizar en forma complementarias las reglas explícitas como simbólicas, incorporando mecanismos adicionales automáticos para el manejo y resolución de inconsistencias e interferencias entre reglas. La estrategia de tejido que implanta MEDIATOR se basa en la generación de una clase de enlace que encapsula las relaciones aspectuales (asociaciones) y la resolución de conflictos (reglas), de esta forma el código (fuente y compilado) tanto de las clases como de los aspectos se mantiene inalterable y no contaminado de esta lógica. Este enfoque tiene varios beneficios: posibilita un mayor reuso del código y facilitar el mantenimiento del código.

El trabajo actual esta dirigido a extender la estructura de la entidad asociación para que permita el uso de condicionales e introducciones de métodos y atributos.

El presente trabajo fue parcialmente financiado por la Universidad Nacional de la Patagonia Austral, Santa Cruz, Argentina y el proyecto PICT 32079 (ANPCYT).

9. Referencias

Casas S., Marcos C., Vanoli V., Reinaga H., Saldivia C., Pryor J., Sierpe L. (2005). "ASTOR: Un Prototipo para la Administración de Conflictos en AspectJ", XIII Encuentro Chileno de Computación, JCC 2.005., Chile.

Casas S., García Perez-Schofield B., Marcos C. (2006) "Gestión de Conflictos entre Aspectos basado en un sistema experto de reglas". XXXII Conferencia Latinoamericana de Informática (CLEI) Chile.

Casas S., García Perez-Schofield B., Marcos C. (2007) “Associations in Conflict”. INFOCOMP – Journal of Computer Science - Federal University of Lavras – Brazil - ISSN: 1807-4545 (to appear)

Charfi A., Riveill M., Blay-Fornarino M., Pinna-Dery A. (2006) “Transparent and Dynamic Aspect Composition”, Workshop on Software Engineering Properties of Languages and Aspects Technologies, VII AOSD – Germany – 2.006

Dijkstra, E.W. (1976). “A Discipline of Programming”, Prentice-Hall, 1976.

Douence R., Fradet P. and Südholt M. (2002). “A Framework for the Detection and Resolution of Aspect Interaction”, In Proceeding of GPCE 2.002, vol. 2487 of LNCS, USA, 2.002, Springe Verlag, pp 173-188.

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, 2.005.

Friedman-Hill E. (2003). “Jess in Action”, Manning Publications, ISBN 1-930110-89-8, 2.003.

Hürsch W., Lopes C. (1995). “Separation of Concerns”. Northeastern University Technical Report NU-CCS-95-03, Boston, 1.995.

Kessler B., Tanter E. (2006). “Analyzing Interactions of Structural Aspects#. Workshop AID in 20th. European Conference on Object-Oriented Programming (ECOOP). France, 2.006.

Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C., Loingtier J., Irwin J. (1997). “Aspect-Oriented Programming”. In Proceedings of ECOOP '97. 1.997.

Kiczales G., Hilsdale E., Hugunin J., Kersten M., Palm J., Griswold W. (2001) “An Overview of AspectJ”. ECOOP 2.001.

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

Nagy I., Bergmans L., Aksit M. (2006) “Composing Aspects at Shared Join Point”. Workshop AID in 20th. European Conference on Object-Oriented Programming (ECOOP). France, 2.006.

Pawlak R., Duchien L., Seinturier L. (2005). “CompAr: Ensuring safe around advice composition”. In FMOODS 2.005, Vol. 3535 of LNCS, pp 163–178, 2.005.

ROOTS (2005). “LogicAJ – A Uniformly Generic and Interference-Aware Aspect Language”; http://roots.iai.uni-bonn.de/researh/logicaj/ , 2.005.

Tanter E., Noye J. (2005). “A Versatile Kernel for Multi-Language AOP”, Proceeding of ACM SIGPLAN/SIGSOFT – Conference on Generative Programming and Component Engineering (GPCE 2.005) LNCS, Springer-Verlag, Estonia, 2.005.

Yu Y., Kienzle J. (2004). “Towards an Efficient Aspect Precedence Model”, Proceeding of the 2.004 Dynamic Aspects Workshop (DAW04), pp 156-167, England, 2.004.