AspectJ vs. MEDIATOR: análisis y comparación del tratamiento de conflictos entre aspectos

Sandra I. Casas
Unidad Académica Río Gallegos, Universidad Nacional de la Patagonia Austral,
Río Gallegos, Argentina, 9400
lis@uarg.unpa.edu.ar

J. Baltasar García Perez-Schofield
Departamento de Informática, Universidad deVigo,
Orense, España, 32004
jbgarcia@uvigo.es

Claudia A. Marcos
Instituto de Sistemas de Tandil, Universidad Nacional del Centro
Tandil, Argentina, 7000
cmarcos@exa.unicen.edu.ar

Abstract.

The occurrence of conflicts among aspects is a consequence of the decomposition of the systems in the aspect-oriented development. This phenomenon is independent of the tools and it requires a special treatment since the activation of certain conflicts can cause non-wanted and unstable behaviors of the software execution. But the conflict treatment is a complex and critic task when the programming tools do not provide appropriate support. In this work two AOP tools are compared: AspectJ and MEDIATOR, bearing in mind the treatment of conflicts.

Keywords: Aspect-Oriented Programming, AspectJ, Conflicts among aspects.

Resumen.

La ocurrencia de conflictos entre aspectos es una consecuencia de la descomposición de los sistemas en el desarrollo orientado a aspectos. Dicho fenómeno es independiente a las herramientas y requiere un especial tratamiento ya que la activación de ciertos conflictos puede provocar comportamientos no deseados, inconsistentes e imprecisión en la ejecución del software. Pero el tratamiento de conflicto es una tarea compleja y crítica si las herramientas de programación no disponen de soporte adecuado. En este trabajo se comparan dos herramientas POA: AspectJ y MEDIATOR en cuanto al tiempo de implementación y mantenimiento teniendo en cuenta el tratamiento de conflictos.

Palabras claves: Programación Orientada a Aspectos, AspectJ, Conflictos entre Aspectos.

1. INTRODUCCIÓN

La Programación Orientada a Aspectos [1] (POA) es un nuevo paradigma de programación que da soporte de implementación al principio de Separación de Concerns [2][3]. La POA propone a la unidad aspecto para implementar a los "crosscutting concerns". Un aspecto esta equipado de dispositivos específicos (puntos de unión, puntos de corte, avisos, introducciones y propiedades) que posibilitan su posterior composición con las unidades funcionales (clases o componentes). La composición es realizada por un proceso denominado "tejido" [4].

Un conflicto entre aspectos puede ocurrir cuando dos o más aspectos compiten por su activación [5]. Algunos autores se refieren al fenómeno en términos de "interacciones" [6] o "interferencias [7]. El problema de los conflictos surge a partir de que ciertos aspectos no son ortogonales y la ejecución descontrolada de los mismos puede producir comportamientos impredecibles, contradictorios, u otros resultados indeseados. En estos casos, el desarrollador debe ser informado y poder administrar estas potenciales situaciones conflictivas. Por estas razones es altamente beneficioso que las herramientas POA (entornos, lenguajes, frameworks, etc.) dispongan de mecanismos y/o dispositivos que automáticamente detecten la presencia de conflictos y proporcionen construcciones específicas y flexibles para la resolución de los mismos. Sin embargo, en la mayoría de las herramientas POA, la identificación, gestión y resolución de conflictos es una tarea compleja y crítica. Lo cual se debe principalmente a dos factores: (i) la detección de conflicto es manual; y (ii) la resolución de conflictos es restringida. La detección de conflictos puede ser una tarea sencilla si la aplicación maneja una reducida cantidad de unidades (clases y aspectos), pero la complejidad de la tarea crece en la medida que aumentan las unidades de codificación de la aplicación. Por otro lado, como ya se ha observado en [8] la mayoría de las herramientas POA ofrecen mecanismos muy limitados para la resolución de conflictos. En general se ofrece un mecanismo que consiste en la fijación de orden, prioridad o precedencia, para lo cual los lenguajes POA proporcionan alguna semántica particular. Por ejemplo, en AspectJ [9] la cláusula "declare precedence" se utiliza para ordenar la ejecución de los avisos de los aspectos en conflicto; en AspectC++ [11] se emplea la cláusula "order"; en DJ Aspect [11] se usa la instrucción "dominates" y en AOP/ST [12] se deben asignar prioridades numéricas a cada aspecto, así los aspectos con alta prioridad se ejecutarán primero. La estructura del lenguaje POA y la forma en que la semántica y sintaxis de resolución de conflictos se conjugan en ciertas ocasiones puede conducir a la re-estructuración (re-diseño y re-codificación) de los aspectos originales, introduciendo mayor complejidad al tratamiento de conflictos e incrementado el costo y esfuerzo de desarrollo o mantenimiento del software.

En este trabajo se analizan los mecanismos y soportes que proporcionan para el tratamiento de conflictos las herramientas POA AspectJ y MEDIATOR [13][14]. Además se discute cómo estas características pueden incidir en el desarrollo del software, para lo cual se proponen las métricas "tiempo de implementación" y "tiempo de mantenimiento", que permiten teóricamente evaluar y comparar estas herramientas.

El presente documento se estructura de la siguiente manera: en las Secciones 2 y 3 se analiza el soporte que brindan AspectJ y MEDIATOR para el tratamiento de conflictos entre aspectos; en la Sección 4 se presentan las métricas tiempo de implementación y tiempo de mantenimiento y su aplicación a AspectJ y MEDIATOR; en la Sección 5 se presentan las conclusiones.

2. ASPECTJ

AspectJ [9] es la herramienta POA más popular y difundida y ha sido considerada en un estado de madurez. Esta extiende de Java incorporando una unidad denominada aspecto. La detección de

conflictos en AspectJ es manual y la resolución de conflictos se realiza mediante la declaración de precedencia. Por ejemplo, “declare precedence A, B” indica que todos los avisos del aspecto A se ejecutarán antes que los del aspecto B. La declaración de precedencia de aspectos presenta severas limitaciones en las siguientes situaciones:

Los aspectos causan más de un conflicto. En la Figura 1 (a), los aspectos A y B causan dos conflictos diferentes. En este caso, si se define la declaración de precedencia “declare precedence: A, B;”, luego resulta imposible, definir “declare precedence: B, A;”. Entonces, si dos aspectos producen más de un conflicto, sobre los cuales es necesario aplicar distintas políticas de orden, la declaración de precedencia resulta insuficiente. La solución en este caso consiste en reestructurar uno de los aspectos. Por ejemplo, si el aspecto B se reestructura en los aspectos B1 y B2, Figura 1 (b), entonces puede establecerse una declaración de precedencia para cada conflicto.

```
  (a)
aspect A {
   before():call(void X.met())
    {........}
   after():execution(void CY.met())
    {.......}
  }
aspect B {
   before():call(void X.met())
    {........}
   after():execution(void CY.met())
    {.......}
  }
```

```
  (b)
aspect B1 {
   declare precedence A,B1;
   before():call(void CX.met())
    {     ........    }
  }

aspect B2 {
   declare precedence B2,A;
   after():execution(void CY.met())
    {     ........    }
  }
```

Figura 1. Los aspectos A y B plantean dos conflictos diferentes.

El conflicto es planteado por distintos pointcuts del mismo aspecto. En la Figura 2 (a) dos pointcuts anónimos del aspecto A plantean un conflicto.

```
  (a)
aspect A {
   before():call (void CX.met())
      {  ........  }
   before ():call (void CX.met())
      {  ........  }
  }
```

```
  (b)
aspect A1 {
   declare precedence A1,A2;
   before():call (void CX.met())
    {     ........    }
  }
aspect A2 {
   after():call (void CX.met())
    {     ........    }
  }
```

Figura 2. Conflicto planteado por el mismo aspecto.

La declaración de precedencia no tiene efectos sobre los avisos de un mismo aspecto. El orden de ejecución de los avisos en conflicto es indefinido. La solución en este caso consiste en reestructurar el aspecto A en dos aspectos. Por ejemplo, sí el aspecto A se re-estructura en los aspectos A1 y A2, Figura 2 (b), es posible establecer la declaración de precedencia entre estos nuevos aspectos.

El orden en que los aspectos deben ser ejecutados depende de una condición del sistema o del contexto. En la Figura 3 (a) se indica que los avisos del aspecto A deben ser ejecutados antes que los avisos del aspecto B si cond es verdadera. Caso contrario, debe ejecutarse después. Esta situación es imposible de implementar en AspectJ debido a que la semántica y sintáxis de la declaración de precedencias no permite condiciones. Además, si se establecen diferentes declaraciones de precedencias que involucren a los mismos aspectos el compilador provoca un error. La solución en este caso consiste en re-estructurar los aspectos A y B, Figura 3 (b), fusionándolos en un aspecto que permita manejar la condición.

```
  (a)
aspect A  {
   if (cond) declare precedence: A, B;
   else declare precedence B,A;
   before():call(void CX.met());
      {  ........  }
  }
aspect B  {
   before():call(void CX.met());
      {  ........  }
  }
```

```
  (b)
aspect AB {
   before():call(void CX.met())
   {  if (cond)
            { metA1(); metB1();}
      else
            { metB1(); metA1(); }
   }
   void metA1()
   {//código del aviso before del aspecto A}
   void metB1()
   {//código del aviso before del aspecto B}
  }
```

Figura 3. Declaración de precedencia y condiciones.

En resumen, el tratamiento de conflictos en AspectJ no sólo se trata de definir declaraciones de precedencia, sino que puede requerir la re-estructuración de los aspectos.

3 MEDIATOR

MEDIATOR es un entorno de programación que permite codificar, compilar y ejecutar aplicaciones, dando soporte a la orientación a aspectos bajo el enfoque denominado Modelo de Asociaciones [13] [14]. MEDIATOR propone incorporar tres nuevas entidades: aspectos, asociaciones y reglas. A continuación se explican los conceptos de aspecto, asociación y regla.

3.1 Aspectos y Asociaciones

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 métodos y atributos. Un aspecto internamente no contiene referencia alguna sobre el componente de funcionalidad básica que cortará transversalmente. En la Figura 4 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 fileLogOperation(String idC, String IdO)
  { // enviar mensaje a fichero   }
}
```

Figura 4. Aspecto Logging

Una asociación es una entidad que define una relación aspectual entre una clase y un aspecto. Una asociación es una unidad que se declara de manera separada y aislada a las demás unidades (aspectos y clases). Una asociación es una unidad de declaración o definición, a diferencia de los aspectos y clases que son unidades de implementación. Por ejemplo, en la Figura 5 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.consoleLogOperation(..). La propiedad priority se utiliza para al resolución de conflicto y solo afecta a la asociación, no al aspecto.

```
association LogAccount
{
  after public void Account.setBalance(float);
  call public void Logging.consoleLogOperation(“Account”, “setBalance”);
  priority = 10;
}
```

Figura 5. Asociación entre el aspecto Logging y la clase Account.

Una asociación es una relación 1-1. Esto significa que sí una clase debe ser relacionada a otros aspectos deben definirse otras asociaciones. De manera similar, si un aspecto corta transversalmente distintas clases, deben definirse distintas asociaciones. Los puntos de unión (“join-points”) se definen por extensión, pero pueden ser definidos por comprensión (cuantificación), mediante el uso de comodines. En este caso MEDIATOR provee mecanismos adicionales automáticos que convierten este tipo de relaciones n-1 en relaciones 1-1.

3.2 Conflictos y Reglas

Un conflicto se produce cuando n asociaciones ($n \geq 2$) definen la misma relación de corte y de aviso sobre el mismo punto de unión. Así una asociación puede participar a lo sumo de un conflicto, aunque un aspecto y/o una clase pueden participar de distintos conflictos. En MEDIATOR el tratamiento de conflictos se hace sobre las asociaciones. Una regla de resolución de conflictos es una entidad que define una acción de resolución para m conflictos de un programa ($m \geq 1$). Una regla de resolución se expresa en dos partes: antecedente y consecuente. La parte del antecedente identifica el conjunto de conflictos que se van a resolver. Y la parte del consecuente 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, 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 6). Por ende, una regla explícita sirve para resolver un único conflicto. En consecuencia el proceso de detección de conflictos debe realizarse previamente. MEDIATOR procesa automáticamente las asociaciones identificando los conflictos.

```
ER :(logAcc1, perAcc3) // antecedente
  =>
  order (perAcc1, logAcc3) //consecuente
```

Figura 6. Regla Explícita.

Las acciones de resolución que se pueden aplicar en una regla explícita pueden ser simples: Orden, Orden Inverso, Opcional, Exclusión y Nulidad. En una regla explícita la acción de resolución puede ser más compleja cuando se aplican más de una acción de resolución simple para resolver el conflicto. Por ejemplo en la Figura 7 se ha definido una regla explícita que combina las acciones de resolución opcional y orden.

```
ER :(logAcc1, perAcc, statAcc3)
  =>
 if (System.time() > 22)
    order (statAcc, perAcc1, logAcc)
 else
    order (logAcc, statAcc, perAcc)
```

Figura 7. Regla explícita y acción de resolución compleja.

Las reglas simbólicas permiten aplicar una acción de resolución a conjuntos de conflictos. En el antecedente se establece una condición que deberán satisfacer los conflictos para que la acción de resolución definida en el consecuente sea aplicada. Luego que se ejecuta el proceso de detección de conflictos MEDIATOR verifica que conflictos son afectados por las reglas simbólicas. Por ejemplo en la Figura 8 se han definido las reglas simbólicas SR1 y SR2.

```
SR1 :  <all>        SR2 :  <class=="Account">
   =>                  =>
 OBP                 OBP
```

Figura 8. Reglas simbólicas.

SR1 aplica la resolución OBP a todos los conflictos existentes, mientras que SR2 solo aplica la resolución OBP a los conflictos sobre la clase Account.

El consecuente asumirá valores concretos de asociaciones luego de la detección de conflictos y la prioridad definida en las asociaciones es el criterio que se utiliza para generar las acciones de resolución. 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. MEDIATOR asegura que la acción de resolución será aplicada en todos los casos. Esta característica da mayor poder a las reglas simbólicas, pero requiere necesariamente de métodos de implementación adicionales más complejos. Las acciones de resolución son:

- OBP: Ordenar las asociaciones en conflicto por prioridad. Ejemplo: OBP (c, b, a);
- IOBP: Ordenar las asociaciones en conflicto en forma inverso a la prioridad. Ejemplo: IOBP(a, b, c);
- ELP: Excluir la asociación en conflicto de menor prioridad. Ejemplo: ELP (a);
- EGP: Excluir la asociación en conflicto de mayor prioridad. Ejemplo: EGP (c);

En resumen, MEDIATOR trata los conflictos de manera individual y aislada. En consecuencia los aspectos no deben ser re-estructurados en ninguna circunstancia. MEDIATOR dispone de mecanismos adicionales para resolver interferencias entre reglas en forma automática. Una interferencia existe cuando dos reglas resuelven el mismo conflicto de distinta forma. Existen dos tipos de interferencia que maneja MEDIATOR: a) entre reglas simbólicas y b) entre regla simbólica y regla explícita. En cuanto al proceso de tejido, los detalles se brindan en [13][14].

4. ANÁLISIS Y COMPARACIÓN

El tiempo que requiere implementar y mantener una aplicación no es fácilmente medible, ya que depende de cuestiones como la experiencia del programador, la herramienta de desarrollo y la complejidad del problema entre otras. Pero a los efectos de poder comparar dos herramientas POA en cuanto a la forma que impacta el soporte que brindan al tratamiento de conflictos sobre el desarrollo se proponen y definen las métricas tiempo de implementación (t_i) y tiempo de mantenimiento (t_m). Estas métricas se plantean para la implementación y mantenimiento software orientado a aspectos y considera el tratamiento de conflictos. A continuación se aplican las métricas propuestas para analizar y comparar en particular a las herramientas POA AspectJ y MEDIATOR.

4.1 Tiempo de Implementación

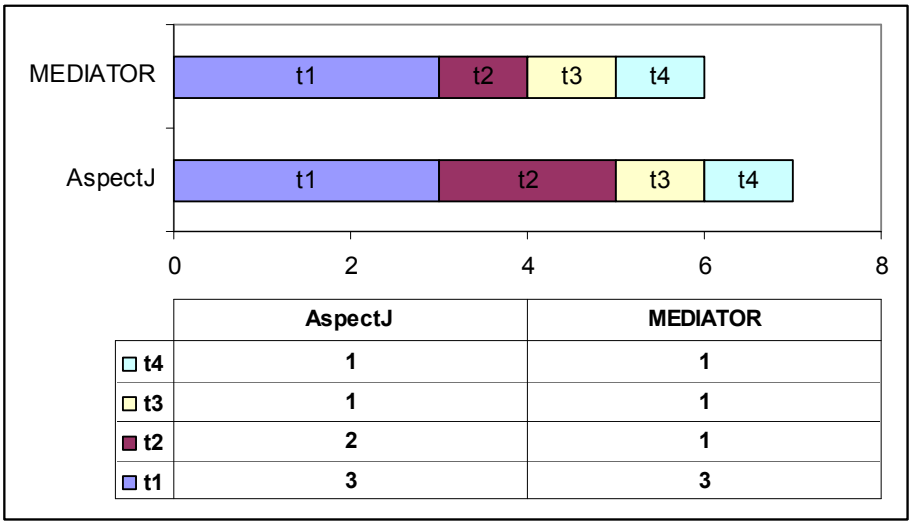
Se define como tiempo de implementación "t_i" al tiempo que insume implementar una aplicación orientada a aspectos completa. La implementación de una aplicación orientada a aspectos consiste en cuatro actividades principales: codificación (de clases, aspectos, asociaciones y/o reglas),

detección de conflictos, resolución de conflictos (declaración de precedencia, definición de reglas y/o re-estructuración de aspectos) y compilación. La realización de cada una de estas actividades, tienen a su vez, un tiempo de realización asociado: t_1, t_2, t_3 y t_4. Así el tiempo de implementación se puede estimar como la sumatoria de los tiempos de las actividades asociadas, es decir: $t_i = t_1 + t_2 + t_3 + t_4$. Los tiempos de cada actividad se miden en unidades de tiempo genéricas. Como punto de partida se supone que todo proceso automático requiere 1 unidad de tiempo, mientras que los procesos manuales requieren: 1 unidad de tiempo la codificación de un aspecto o una clase, 1 unidad de tiempo la inspección de una clase o un aspecto y 1 unidad de tiempo la resolución de un conflicto particular.

En este punto se puede asumir que t_1 (codificación de aspectos y clases) y t_4 (compilación) requieren un tiempo y esfuerzo similar. Las principales diferencias están dadas en t_2 y t_3. Utilizando MEDIATOR $t_2 = 1$ siempre, ya que es un proceso automático. En cambio $t_2 > 1$ siempre en AspectJ, ya que es un proceso absolutamente manual, incluso si no existen conflictos. El desarrollador debe analizar por lo menos todos los aspectos codificados, y por cada aspecto deben inspeccionarse todos los puntos de cortes y avisos definidos. Pero t_2 puede aumentar considerablemente en AspectJ, si los puntos de unión ("join-points") han sido definidos por comprensión utilizando comodines, ya que deberán analizarse también la estructura de las clases para poder determinar fehacientemente la existencia de conflictos. Respecto de t_3, en el caso de MEDIATOR se puede asumir que la especificación de una regla (explícita o simbólica) insume 1 unidad de tiempo En el caso de AspectJ también se puede asumir que la especificación de una declaración de precedencia, insume 1 unidad de tiempo. Visto de este modo, parecería que t_3 tiene un tiempo similar en ambos enfoques, pero no es así. En principio, una regla simbólica equivale a n declaraciones de precedencia, aquí se origina una primera diferencia en favor de MEDIATOR. Pero además, se debe tener en cuenta que t_3 puede aumentar considerablemente en AspectJ, si el conflicto no se puede resolver con la declaración de precedencia y es necesario re-diseñar y re-codificar los aspectos, lo cual implica un tiempo adicional, que en MEDIATOR no es necesario.

A continuación se proporcionan dos análisis para el cálculo de la métrica ti. El primero se refiere al análisis del mejor caso y el segundo al análisis del peor caso. El estudio se hace sobre la suposición que la aplicación orientada a aspectos se compone de 1 clase y 2 aspectos, sobre los cuales se plantean dos conflictos.

En cuanto al análisis del mejor caso, la tabla de la Figura 9 refleja los tiempos de cada actividad.

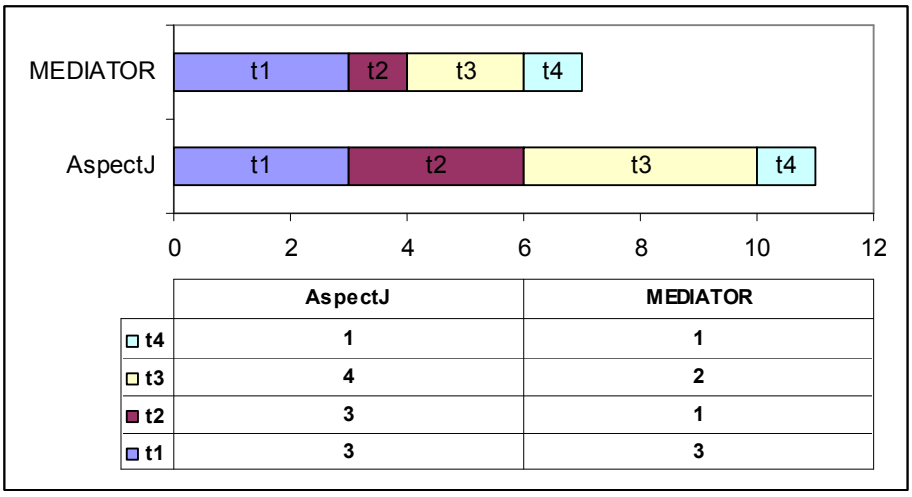


	AspectJ	MEDIATOR
t4	1	1
t3	1	1
t2	2	1
t1	3	3

Figura 9. Tiempo de Implementación (t_i): Mejor Caso.

Aquí se asigna 1 unidad de tiempo por cada unidad (clase o aspecto) codificada, lo que resulta en t_1 = 3 en ambos casos. t_2 = 2 para AspectJ ya que se deben inspeccionar los dos aspectos para detectar la existencia de todos los posibles conflictos, mientras que t_2 = 1 en MEDIATOR, ya que es un proceso automático. t_3 = 1 en ambos casos, ya que se asume que los conflictos han sido resuelto con una declaración de precedencia (AspectJ) o una regla (MEDIATOR). t_4 = 1 ya que se asume que requiere una unidad de tiempo (siendo un proceso automático en ambos sistemas). Finalmente t_i = 7 en AspectJ y t_i = 6 en MEDIATOR, para el análisis del mejor caso. Como se ilustra en el gráfico de la Figura 9 el t_i de AspectJ es un 16% más costoso que el t_i de MEDIATOR.

En cuanto al análisis del peor caso, éste se plantearía en la situación que para la identificación de conflictos es necesaria analizar además de los aspectos, la estructura de las clases (los puntos de unión han sido definidos por comprensión mediante el uso de comodines) y que la resolución de conflictos en AspectJ no se puede realizar mediante la simple definición de la declaración de precedencia y se requiere en consecuencia recodificar al menos uno de los aspectos en conflicto. La tabla de la Figura 10 refleja el tiempo insumido por cada actividad. Al igual que en el caso anterior t_1 = 3 en ambos casos. t_2 = 3 para AspectJ ya que se deben inspeccionar los dos aspectos y la estructura de la clase, mientras que t_2 = 1 en MEDIATOR, ya que es un proceso automático. t_3 = 4 en AspectJ por que es necesario la re-codificación de uno de los aspectos (el aspecto se dividió en dos aspectos: dos unidades de tiempo) y sobre éstos se aplicó otra declaración de precedencia. t_3 = 2 ya que cada para cada conflicto se definió 1 regla diferente. t_4 = 1 al igual que en el caso anterior para ambos casos. Finalmente t_i = 11 en AspectJ y t_i = 7 en MEDIATOR, para el análisis del peor caso. Como se ilustra en el gráfico de la Figura 10, el t_i de AspectJ es un 57% superior que el t_i de MEDIATOR.



	AspectJ	MEDIATOR
t4	1	1
t3	4	2
t2	3	1
t1	3	3

Figura 10. Tiempo de Implementación (t_i): Peor Caso.

En principio estos análisis revelan que la detección y resolución de conflictos entre aspectos son actividades que pueden afectar severamente el tiempo de implementación del software. Luego de los análisis realizados se desprende que la implementación con AspectJ requiere entre un 16% y 57% más de tiempo que MEDIATOR. Es decir, en promedio requiere un 36,5% más de tiempo.

4.2 Tiempo de Mantenimiento

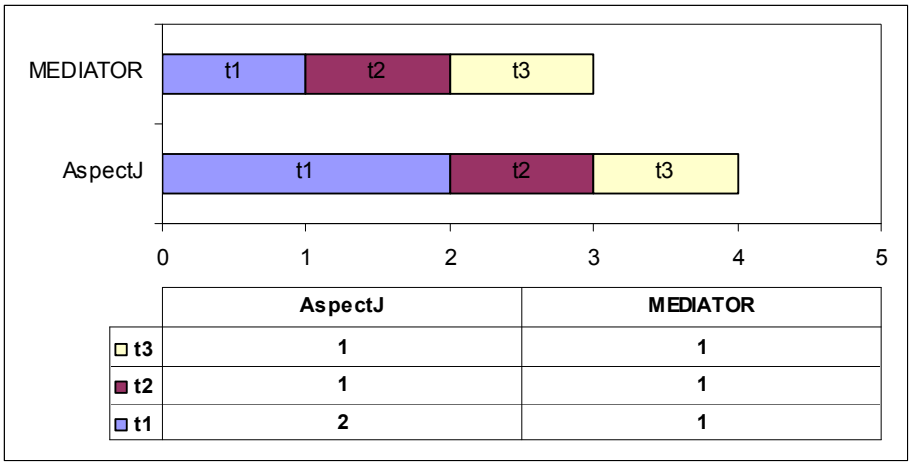
Se define como tiempo de mantenimiento "t_m" al tiempo que insume modificar la política de resolución de conflictos en una aplicación orientada a aspectos implantada. La modificación de la política de resolución de conflictos en una aplicación orientada a aspectos se puede descomponer en tres actividades principales: localizar la actual política de resolución, re-definir la política de

resolución y compilación. La realización de cada una de estas actividades, tienen a su vez, un tiempo de realización asociado: t_1, t_2 y t_3. Así el tiempo de mantenimiento se puede estimar como la sumatoria de los tiempos de las actividades asociadas, es decir: $t_m = t_1 + t_2 + t_3$. Los tiempos de cada actividad se miden en unidades de tiempo genéricas. Como punto de partida se supone que todo proceso automático requiere 1 unidad de tiempo, mientras que los procesos manuales requieren: 1 unidad de tiempo la codificación de un aspecto o una clase, 1 unidad de tiempo la inspección de una clase o un aspecto y 1 unidad de tiempo la resolución de un conflicto particular.

En este caso, t_1 resulta algo más costoso en AspectJ que en MEDIATOR ya que deben inspeccionarse todos los aspectos para localizar las declaraciones de precedencia definidas y comprender sobre que avisos en particular están actuando. Por el contrario, en MEDIATOR sólo deben inspeccionarse las reglas. Para agilizar y hacer más eficiente esta tarea en AspectJ sería de gran utilidad documentar las declaraciones de precedencia definidas, lo cual es una actividad adicional que en MEDIATOR no es necesario realizar, ya que las reglas de por sí sirven cómo documentación de las políticas de resolución de conflictos aplicadas o incluso la edición de la clase de enlace. t_2 se comporta de la misma manera que t_3 en el análisis de t_i, es decir t_2 puede tener un tiempo similar pero con grandes posibilidades de aumentar considerablemente en AspectJ. Respecto de t_3, en AspectJ debe recompilarse toda la aplicación (clases y aspectos) para que los nuevos cambios tengan efecto, mientras que en MEDIATOR sólo debe recompilarse la clase de enlace.

A continuación se proporcionan dos experimentos teóricos para el cálculo de la métrica t_m. El primero se refiere al análisis del mejor caso y el segundo al análisis del peor caso. El estudio se hace sobre la suposición que la aplicación orientada a aspectos se compone de 1 clase y 2 aspectos, sobre los cuales se plantean dos conflictos (cada conflicto esta formado por dos aspectos) y estos han sido resueltos mediante la definición de una declaración de precedencia en el caso de AspectJ y mediante la definición de una regla en el caso de MEDIATOR.

En cuanto al análisis del mejor caso, la Figura 11 refleja los tiempos de cada actividad.



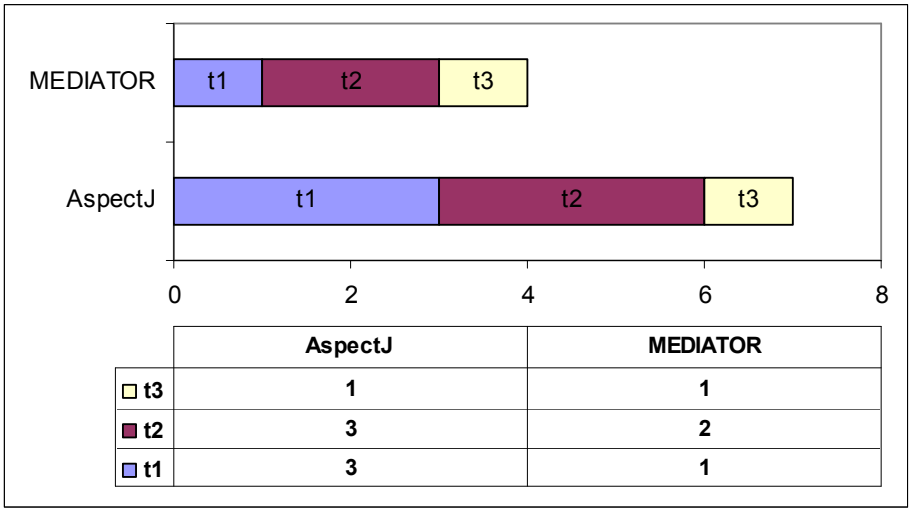
	AspectJ	MEDIATOR
t3	1	1
t2	1	1
t1	2	1

Figura 11. Tiempo de Mantenimiento (t_m): Mejor Caso.

Aquí $t_1 = 2$ para AspectJ por que deben inspeccionarse los dos aspectos para localizar la declaración de precedencia, mientras que $t_1 = 1$ en MEDIATOR ya que solo debe inspeccionarse la regla para determinar sobre que asociaciones y aspectos actúa. $t_2 = 1$ para ambos casos ya que una vez ubicada la declaración de precedencia su modificación requiere un tiempo similar a modificar una regla. $t_3 = 1$ ya que se asume que requiere una unidad de tiempo (siendo un proceso automático en ambos sistemas). Finalmente $t_m = 4$ en AspectJ y $t_m = 3$ en MEDIATOR, para el análisis del

mejor caso. Como se ilustra en el gráfico de la Figura 11, el t_m de AspectJ es un 33% más costoso en términos de tiempo que el tm de MEDIATOR, para el análisis del mejor caso.

En cuanto al análisis del peor caso (Figura 12), aquí $t_1 = 3$ en AspectJ ya que para localizar la declaración de precedencia y comprender sus efectos se requiere no sólo inspeccionar los aspectos sino además examinar la estructura de las clases (los puntos de unión han sido definidos por comprensión mediante el uso de comodines), mientras que $t_1 = 1$ en MEDIATOR ya que la definición de la regla no sólo declara el método de resolución aplicado sino además sobre que asociaciones y aspectos esta actuando (también puede editarse la clase de enlace con el mismo propósito). $t_2 = 3$ en AspectJ por que es necesario por cada conflicto la re-codificación de uno de los aspectos (el aspecto se dividió en dos aspectos: dos unidades de tiempo) y sobre éstos se aplicó otra declaración de precedencia. En MEDIATOR $t_2 = 2$ ya que para cada conflicto se definió 1 regla diferente. $t_3 = 1$ al igual que en el caso anterior para ambos casos. Finalmente $t_m = 7$ en AspectJ y $t_m = 4$ en MEDIATOR, para el análisis del peor caso. Como se ilustra en el gráfico de la Figura 12, el t_m de AspectJ prácticamente duplica el t_m de MEDIATOR, ya que requiere un 75% más de tiempo.



	AspectJ	MEDIATOR
t3	1	1
t2	3	2
t1	3	1

Figura 12. Tiempo de Mantenimiento (t_m): Peor Caso.

En principio estos experimentos revelan que la identificación de las políticas de resolución de conflictos entre aspectos aplicadas y la modificación de las mismas son actividades realmente críticas del mantenimiento del software. Luego de los análisis realizados se desprende que el mantenimiento con AspectJ requiere entre un 33% y 75% más de tiempo que MEDIATOR. Es decir, en promedio requiere un 50% más de tiempo.

5. CONCLUSIONES

El tratamiento de conflictos entre aspectos es crítico si las herramientas POA no cuentan con dispositivos flexibles que eviten la necesidad de realizar operaciones manuales y la re-estructuración del código de los aspectos. Este trabajo ha propuesto analizar las herramientas POA AspectJ y MEDIATOR, desde el soporte al tratamiento de conflictos que brindan. Para ello se ha definido la métricas t_i que estima el tiempo de implementación y la métrica t_m que estima el tiempo de mantenimiento. MEDIATOR ha demostrado ser una herramienta más eficiente en ambos casos, esto se debe a que dispone de mecanismos y dispositivos específicos y flexibles para el tratamiento de conflictos entre aspectos.

Por otro lado, se debe destacar que en general las herramientas POA se comparan y evalúan en cuanto al tiempo de ejecución, como se hace en [15]. El parecer de los autores de éste trabajo, es que la performance o rendimiento medido en términos de tiempo de ejecución es importante pero no suficiente. Las herramientas POA deben además ser analizadas desde otros factores, por ejemplo el que se propone en este trabajo, u otros que permitan por ejemplo analizar los mecanismos de reutilización soportados.

Referencias

[1] Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C., Loingtier J., Irwin J. "Aspect-Oriented Programming". In Proceedings of ECOOP (1.997).

[2] Hürsch W., Lopes C. "Separation of Concerns". Northeastern University, TR NU-CCS-95-03, Boston (1.995).

[3] Dijkstra E. "A Discipline of Programming", Prentice-Hall (1.976).

[4] Piveta E., Zancanela L. "Aspect Weaving Strategies". Journal of Universal Computer Science, vol.9, no. 8, (2.003).

[5] Pryor J., Diaz Pace A., Campo M. "Reflection on Separation of Concerns". RITA. Vol.9. Num.1 (2.002).

[6] Tanter E., Noye J. "A Versatile Kernel for Multi-Language AOP". Proceeding of ACM SIGPLAN/SIGSOFT – (GPCE) LNCS, Springer-Verlag, Estonia, (2.005).

[7] Katz S., Rashid A. "From Aspectual Requirements to Proof Obligations for Aspect-Oriented Systems". International Conference on Requirements Engineering (RE), Japon, IEEE Computer Society Press. Pp 48-57 (2.004).

[8] Casas S., Reinaga H., Sierpe L., Vanoli V., Saldivia C., Prior J. "Clasificación y Resolución de Conflictos entre Aspectos". VII Workshop de Investigadores en Ciencias de la Computación – Argentina (2.005).

[9] Kiczales G., Hilsdale E., Hugunin J., Kersten M., Palm J., Griswold W. "An Overview of AspectJ". ECOOP (2.001).

[10] Gal A., Schroder W., Spinczyk O. "AspectC++: Language Proposal and Prototype Implementation". ACM International Conference Series Proc. of the 40th International Conference on Tools Pacific. Vol.10. Australia. (2.002).

[11] Pryor J., Bastán N., Campo M. "A Reflective Approach to Support Aspect Oriented Programming in Java". In Proc. of the ASSE. 29 JAIIO. Argentina. (2.000).

[12] Boellert, K. "On Weaving Aspects". Proc. of the AOP Workshop at ECOOP (1.999).

[13] Casas S., Perez-Schofield B., Marcos C. "Associations in Conflict". INFOCOMP – Journal of Computer Science - Federal University of Lavras Brazil - vol. 6 – num. 2 - ISSN: 1807-4545. (2.007)

[14] Casas S., Perez-Schofield B., Marcos C. "Modelo de Asociaciones: un enfoque para el tratamiento de conflictos entre aspectos". I Latin American Workshop on Aspect-Oriented Software Development. João Pessoa, Brasil. (2.007).

[15] AOP Benchmark. AOP Benchmark - AspectWerkz - Confluence http://docs.codehaus.org/display/AW/AOP+Benchmark

El presente trabajo fue parcialmente financiado por la Universidad Nacional de la Patagonia Austral, Santa Cruz, Argentina.