Niveles de Riesgo de Conflictos en AspectJ: una propuesta basada en el pasaje del contexto

Sandra I. Casas[1] y Claudia Marcos[2]

[1]Unidad Académica Río Gallegos, Universidad Nacional de la Patagonia Austral
Río Gallegos, Argentina, 9400

[2]Instituto de Sistemas de Tandil, Universidad Nacional del Centro
Tandil, Argentina, 7000

lis@uarg.unpa.edu.ar, cmarcos@exa.unicen.edu.ar

Abstract. *En el desarrollo de software orientado a aspectos, dos o más aspectos pueden plantear un conflicto. En ciertos casos esta interacción provoca un comportamiento impredecible e inestable del software. Este trabajo propone un método de detección y estimación de niveles de riesgo de conflictos específica para AspectJ. El pre-procesamiento del código fuente de los aspectos facilita estas operaciones. La estimación de los riesgos esta basado en el uso y modo del contexto en los aspectos. De esta forma los riesgos se clasifican en distintos niveles.*

1. Introducción

La Programación Orientada a Aspectos [Kiczales et al. 1997] (POA) es un nuevo enfoque para el desarrollo de software que proporciona mecanismos y abstracciones para la implementación de los crosscutting concerns (requerimientos no funcionales y transversales) de manera separa y aislada. La orientación a aspectos, es una técnica que permite aplicar el principio de Separación de Concerns (SoC) [Dijkstra 1976][Hursch and Lopes 1995].

La POA propone la encapsulación de los crosscuting concerns en unidades de código denominadas aspectos. Los aspectos incluyen mecanismos sintácticos y semánticos que permiten su posterior composición con el código de funcionalidad básica. La composición es realizada por un proceso denominado tejido, que puede ser estático o dinámico [Piveta and Zancanela 2003]. Los dispositivos de invocación y extensión aspectual [Hanenberg and Unland 2001] son implícitos, permitiendo que los aspectos se diseminen y mezclen transversalmente en los módulos de funcionalidad básica.

Desde esta perspectiva dos o más aspectos pueden cortar transversalmente el mismo módulo de funcionalidad básica. Esta situación se conoce como "*conflicto entre aspectos"*, también denominado "*interacción*" [Tanter and Noye 2005] o "*interferencia*" [Katz and Rashid 2004]. En ciertos casos estas interacciones, pueden provocar un comportamiento impredecible del software. De esta manera, surgen los

conflictos entre aspectos y la necesidad de construir mecanismos y estrategias para su adecuado tratamiento.

El presente trabajo aborda la problemática de conflictos entre aspectos y propone un método de detección y estimación de riesgos de conflictos específica para AspectJ [AspectJ 2008][Kiczales et al. 2001]. El método esta basado en el análisis de la estructura de los pointcuts y las primitivas que permiten capturar el contexto de ejecución de los join-points. La detección de conflictos y la posterior estimación de los niveles de riesgos de los mismos se realizan mediante el pre-procesamiento del código de los aspectos.

El presente trabajo se estructura de la siguiente manera: en la Sección 2 se describe brevemente AspectJ y los conflictos entre aspectos en dicha herramienta; en la Sección 3 se explica el método aplicado para la detección de conflictos; en la Sección 4 se expone los niveles de riesgo de conflictos y el procedimiento de cálculo de los mismos; en la Sección 5 se presentan los trabajos relacionados y en la Sección 6 las conclusiones.

2. AspectJ y Conflictos

Un aspecto constituye en AspectJ [AspectJ 2008][Kiczales et al. 2001], la unidad modular o constructor que representa un tipo entrecruzado al cortar las clases, interfaces y a otros aspectos mejorando la SoC. Un aspecto es un conjunto de pointcuts, advices, introducciones, métodos y atributos. Para el presente trabajo, interesan en particular los constructores denominados crosscutting dinámicos. Estos modifican el comportamiento de la ejecución de un programa, en AspectJ se denominan *pointcuts* y *advices*. AspectJ provee un conjunto de cortes primitivos (*call, execution, set, get, handler, etc.*) que indican la semántica del corte (captura) del join-point y determinan la estructura de los pointcuts. Los pointcuts se asociacian a los advices que pueden ser: *before, after y around*. Un pointcut no solo captura la ejecución de un join-point, también puede exponer parte de la ejecución del contexto del join-point. Los valores expuestos por el pointcut pueden ser utilizados en el cuerpo del advice, mediante pasaje de parámetros. Las primitivas para exponer el contexto en AspectJ son: *target, this y args*. En la Figura 1 se capturan todas las llamadas al método *m()* de la clase *A*. La primitiva target permite capturar en la referencia *obj*, el objeto receptor del mensaje enunciado en el join-point. *obj* puede ser manipulado como se desee en el cuerpo del advice. La primitiva *this* permite captura el objeto actual (esto es cuando el join-point comienza su ejecución) y la primitiva *args* permite capturar los argumentos del join-points.

```
before (A obj) : call (void A.m() && target (obj));
```

Figura 1. Advice y pointcut en AspectJ.

Un conflicto puede ocurrir cuando dos o más aspectos compiten por su activación [Pryor et al. 2002]. Al analizar las potenciales situaciones conflictivas que pueden ocurrir en particular en AspectJ, se establece que existe un potencial conflicto sí dos o más aspectos definen pointcuts, cuyos cortes coinciden en tipo de corte y join-points, los cuales además están asociados al mismo tipo de advice. La Figura 2 presenta los aspectos A y B los cuales están en conflicto.

```
aspect A {                          aspect B {
  before():call(void A.m());          before():call(void A.m());
  {     ........     }                {      ........     }
}                                   }
```

Figura 2. Conflicto de Semejanza Total entre los aspectos A y B.

3. Detección de Conflictos

La detección de conflictos tiene por objetivo la identificación de conflictos y la generación de información útil para su análisis y resolución. Este proceso 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 componentes de la aplicación. En estos casos, el desarrollador debe ser informado y poder controlar estas potenciales situaciones para determinar la ejecución deseada de acuerdo al tipo de conflicto y/o el dominio de la aplicación, determinando las prioridades y políticas de activación de los aspectos. En AspectJ la declaración de los pointcuts se estructura en 2 partes:

```
                  (a)                    (b)
call(void A.m(..))   &&   target(A)  &&  args(..)
```

Donde: (a) captura del join-point y (b) captura del contexto del join-point. La información de (a) se utiliza para detectar el conflicto y la información de (b) para estimar el nivel de riesgo más adelante. Esta información puede ser obtenida mediante pre-procesamiento del código de los aspectos. Los pointcuts de los aspectos son mapeados en una tabla que contiene la siguiente información:

```
PointcutTable = {(ASPECT, POINTCUT, PRIMITIVE-CUT, JOIN-POINT, ADVICE, this,
                          target, args, R/W)}
```

En la Figura 3 se indica como ejemplo el mapeo del aspecto “Asp” en la tabla dinámica.

```
aspect Asp {
  pointcut p1 (int i): call (void A.methodX(int) && args (i));
  pointcut p2 (A obj, int i): execution void A.methodY(int)
                              && this(obj) && arg(i));
  before(int i) : p1(i)
  {   …  }
  void around(A obj, int i) : p2(i)
  {  …  }
  after() :  set (int A.atributeX());
  {  …. }
}
```

…									
Asp	p1	call	A	methodX	before	false	false	true	R
Asp	p2	execution	A	methodY	around	true	false	true	W
Asp		set	A	atributeX	after	false	false	false	-
…									
…									

Figura 3. Mapeo del aspecto Asp en la tabla de pointcuts.

En este caso el mapeo crea 3 entradas en la tabla dinámica, uno por cada pointcut (p1, p2 y el pointcut anónimo). Aquí los pointcuts se componen sólo de un corte primitivo (call, execution, set) sobre join-points definidos por extensión (void A.metodoX(), void A.metodoY(), int A.atributoX). Las informacion registrada en las últimas cuatro columnas se genera cuando se mapea el aspecto, pero será empleada para la estimacion del riesgo. La última columna de la tabla describe el uso del contexto en el cuerpo del advice. Este será "R" (read) si el contexto sólo es leído y será "W" si el contexto es modificado.

En AspectJ un pointcut se puede definir por la composición de varios cortes primitivos (utilizando los constuctores "&&", "||" y "!"), además los join-points se pueden definir por comprensión, mediante el uso de comodines ("*" y ".."). En estos casos, el mapeo de un aspecto no será tan lineal como el ejemplo, es decir un join-point puede producir más de una entrada en la tabla de pointcuts. Previamente al mapeo, cada pointcut es analizado y se determina el conjunto de entradas que produce en la tabla. Luego que todos los pointcuts de todos los aspectos han sido mapeados en la tabla, un algoritmo verifica la existencia de conflictos y a continuación se procede a estimar el nivel de riesgo de cada uno.

4. Niveles de Riesgos de Conflictos

Las operaciones de detección y resolución de conflictos adquieren mayor importancia cuando el conflicto produce un comportamiento anómalo del software. Las posibilidades de que un conflicto provoque un comportamiento anómalo aumentan cuando los pointcuts exponen el contexto de los join-points, para que éstos sean manipulados en los advice. Es decir, si los aspectos modifican el estado del contexto de manera no coordinada y/o desordenada durante la ejecución del software, ciertas variables pueden asumir un estado impredecible e inestable durante la ejecución.

La información de la tabla generada para la detección de conflictos provee suficiente información para estimar los riesgos asociados a cada conflicto. Como se explicó anteriormente, en AspectJ existen tres formas de exponer el contexto (target, this y args), en principio se mide el uso del contexto por cada elemento que esta en conflicto, de acuerdo a la Tabla 1.

Tabla 1. Usos del Contexto.

Ejemplo	Uso de contexto	Descripción
`pointcut p () : call (A.m(int));`	N/C	No se usa el contexto
`pointcut p (A ref) : call (A.m(int)) && target (ref);`	Parcial	Usa una parte del contexto
`pointcut p (A ref) : execution (A.m(int)) && this (ref);`	Parcial	Usa una parte del contexto
`pointcut p (int n) : call (A.m(int)) && args (n);`	Parcial	Usa una parte del contexto
`pointcut p (A ref, int n) : call (A.m(int)) && target (ref) && arg (n);`	Total	Usa todo el contexto
`pointcut p (A ref, int n) : execution (A.m(int)) && this (ref) && arg (n) ;`	Total	Usa todo el contexto

Si el uso del contexto es parcial o total, además se requiere conocer el modo de uso. Este será de lectura (R) o escritura (W). Luego, de acuerdo al modo en que se emplee el contexto, los niveles de riesgo de un conflicto pueden ser: nulo, bajo, medio o alto. Las siguientes situaciones determinan el nivel de riesgo de un conflicto entre aspecto en AspectJ:

Nivel de Riesgo Nulo: El nivel nulo implica que el conflicto no conlleva ningún riesgo y por ende no requiere una resolución. Este nivel de riesgo se produce cuando al menos uno de los aspectos no usa el contexto. En la Figura 4 se esquematiza al conflicto mediante un árbol and-or, en el cual las ramas representan cada uno de los aspectos y los posibles usos y modos del contexto que conducen a este nivel de riesgo.

Nivel de Riesgo Bajo: Un conflicto tiene nivel de riesgo bajo, cuando ambos aspectos usan el contexto para lectura. En la Figura 5 se esquematiza el árbol and-or para esta situación. Este nivel de riesgo denota que a pesar de que la ejecución de los aspectos requiera una secuencialidad sí esta no se respetará no ocasionará un comportamiento inestable del software.

Nivel de Riesgo Medio: A un conflicto le corresponde nivel de riesgo medio si uno *de* los aspectos usa el contexto para lectura y el otro aspecto lo usa para escritura, como se representa en la Figura 6. En este nivel de riesgo, la ejecución de los aspectos requiere una secuencialidad preestablecida que debe ser respetará, en consecuencia el desarrollador debe definir una acción de resolución especifica.

Nivel de Riesgo Alto: El nivel de riesgo alto, es el más peligroso y requiere una intervención del desarrollador, ya que en este caso los aspectos en conflicto modifican el contexto, como se grafica en la Figura 7. Dado que diferentes órdenes de ejecución pueden provocar un comportamiento inestable del sistema, este nivel de conflicto requiere resolución explícita.

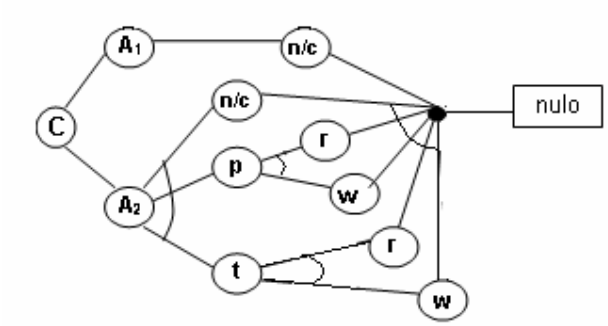

Figura 4. Arbol and-or para conflictos de nivel nulo.

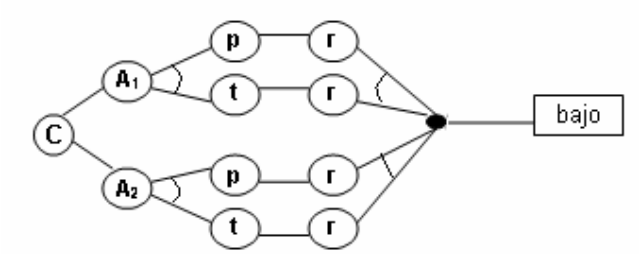

Figura 5. Arbol and-or para conflictos de nivel bajo.

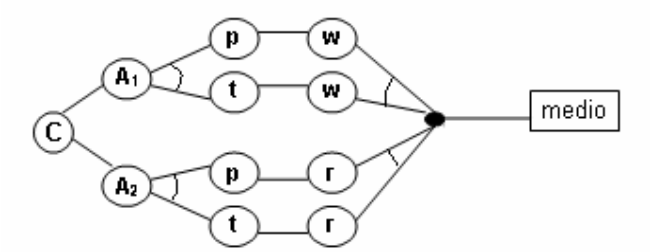

Figura 6. Arbol and-or para conflictos de nivel medio.

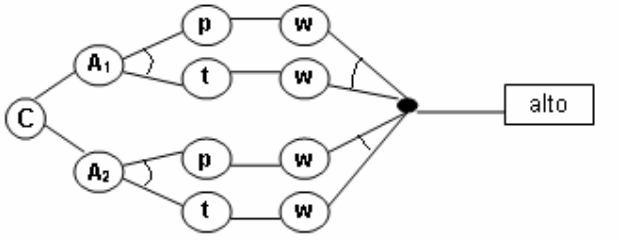

Figura 7. Arbol and-or para conflictos de nivel alto.

Los casos analizados se mapean en una tabla de doble entrada que para dos aspectos en conflicto presenta 25 posibilidades, las cuales se representan en la Tabla 2.

Tabla 2. Niveles de riesgos.

			A_2				
			N/U	PARCIAL		TOTAL	
				W	R	W	R
A_1	N/U		NULO	NULO	NULO	NULO	NULO
	PARCIAL	R	NULO	MEDIO	BAJO	MEDIO	BAJO
		W	NULO	ALTO	MEDIO	ALTO	MEDIO
	TOTAL	R	NULO	MEDIO	BAJO	MEDIO	BAJO
		W	NULO	ALTO	MEDIO	ALTO	MEDIO

Entonces, luego que se detectan los conflictos, se aplica la Tabla 2 para estimar el nivel de riesgo de los mismos.

4.1 Conflictos N

Un conflicto N ocurre cuando *n* aspectos plantean el conflicto y $n > 2$. En estos casos la detección de conflictos se realiza de la manera indicada pero el cálculo de riesgos requiere de un proceso de reducción previo. A continuación se describe el algoritmo de reducción: aquí **n** es la cantidad de aspectos en conflicto.

```
Paso 1: r ← aspectos del conflicto cuyo uso del contexto es N/C
        n ← n – r
        Si n ≤ 1 nivel de riesgo ← NULO – Fin
        Si n = 2 nivel de riesgo ← aplicar Tabla 2– Fin
        Si n > 2 hacer Paso 2

Paso 2: r ← aspectos del conflicto cuyo uso del contexto es R
        n ← n – r
        Si n = 0 nivel de riesgo ← BAJO – Fin
        Si n = 1 nivel de riesgo ← MEDIO – Fin
        Si n > 1 nivel de riesgo ← ALTO – Fin
```

A continuación se presentan dos ejemplos sencillos del proceso de reducción. El primero (Figura 8) presenta un conflicto planteado por tres aspectos en el cual uno de ellos no usa el contexto. Luego de aplicar el paso 1, se aplica la tabla 2 y el conflicto resulta tener un nivel de riesgo MEDIO.

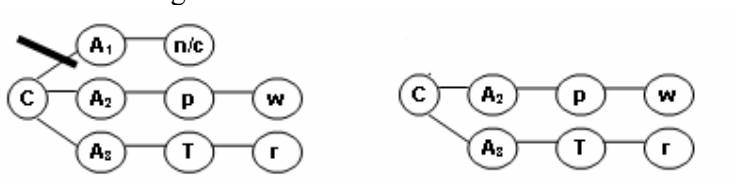

Figura 8. Reducción en Conflictos N.

El segundo ejemplo (Figura 9), también consiste en tres aspectos, luego de aplicar el paso 1, no encuentra resolución por lo que requiere aplicar el paso 2. Luego resulta que el conflicto corresponde al nivel de riesgo MEDIO.

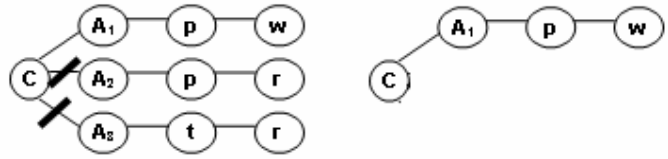

Figura 9. Reducción en Conflictos N.

5. Trabajos Relacionados

El trabajo de investigación en cuanto a conflictos entre aspectos, se ha dirigido principalmente en los procesos de detección y resolución de conflictos, menor atención ha recibido la estimación de los mismos.

[Duence et al. 2002] 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 [ROOTS 2005] provee análisis de las interferencias aspecto-aspecto para AspectJ que incluye capacidades para: (a) identificar una clase bien definida de interferencias, (b) determinar el orden de ejecución libre de interferencia (c) determinar el algoritmo de tejido más conveniente para un conjunto de aspectos dado. El análisis de interferencias es independiente de los programas base a los que los aspectos se refieren (solo los aspectos son necesarios para el análisis) e independiente de anotaciones especiales del analizador de aspectos. Este trabajo solo enfoque el problema de la detección de conflictos.

Astor [Casas et al. 2005] es un prototipo que propone una serie de mecanismos y estrategias para mejorar el tratamiento de conflictos en AspectJ. Los mismos se soportan mediante la adición de un componente Administrador de Conflictos que cumple principalmente con las funciones de detectar automáticamente conflictos y aplicar estrategias de resolución más amplias que las que AspectJ tiene por defecto, en forma semiautomática. 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 [Pryor and Marcos 2003].

La implementación del prototipo esta basada en el pre-procesamiento de código AspectJ, siendo además éste el único requisito para su uso.

Programme Slicing 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. [Monga et al. 2003] Este trabajo solo da cobertura al problema de la detección.

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.

En [Kessler and Tanter 2006] 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 [Tessier et al. 2004]. La identificación de conflictos se realiza a través de información generada a partir del diagrama de clase de UML extendido, que incluye aspectos y sus conexiones con las clases. Información tal como pointcuts, join-points, advice es mapeada en tablas que permiten que un algoritmo identifique todas las posibles relaciones entre las clases y aspectos. A partir del tipo de relación se obtiene una predicción del impacto del conflicto y se genera una recomendación. El propósito de los autores es identificar las interacciones entre aspectos en la fase de modelado, y proporcionar un método formal que mediante refinamientos sucesivos permita detectar los conflictos. El objetivo principal es lograr la detección de conflictos lo antes posible (detección temprana de conflictos) y ofrecer cierto nivel de predicción del impacto generado por la inserción de nuevos aspectos. Las inconvenientes de este trabajo son: (i) la información utilizada para la predicción del impacto es manual, en las tablas se agrega por cada pointcuts-advice el tipo de operación (modificación o acceso), (ii) el modelo genera información redundantes, que para un ejemplo simple no resulta confusa, pero en un caso real complica el análisis del desarrollador y (iii) la información de diseño no es suficiente para la detección y requiere de refinamientos sucesivos que derivan en el análisis estático en la implementación.

6. Conclusiones

El presente trabajo propone un método de detección y estimación de niveles riesgos de conflictos entre aspectos, específico para AspectJ. El código fuente de los aspectos es pre-procesado para obtener información de los pointcuts y advice que es empleada para la detección y estimación de riesgo de los conflictos. Los niveles de riesgo de los conflictos se determinan de acuerdo al uso del contexto de los aspectos en conflictos, y estos se clasifican según su estructura en n/c, parcial o total. Se tiene en cuenta además el modo en que el contexto es empleado en los advice, si el mismo es de lectura o escritura. Según esta clasificación los conflictos presentan diferentes niveles de riesgo: nulo, bajo, medio o alto. Esta información ayuda al desarrollador en la resolución de conflictos.

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

Referencias

AspectJ (2008) http://aspectj.org/.

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.

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.

Dijkstra, E. (1976) "A Discipline of Programming", Prentice-Hall.

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.

Hanenberg, S., Unland, R. (2001) "Concerning AOP and Inheritance". Proceedings of Workshop Aspect-Orientation, Paderborn.

Hursch, W., Lopes, C. (1995) "Separation of Concern". TR NU-CCS-95-03, Northeastern University.

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

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

Kiczales, G., Lamping, L., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J. (1997) "Aspect-Oriented Programming". In Proceedings ECOOP'97 – Object-Oriented Programming, 11th European Conference, Jyväskylä Finland, Springer-Verlang.

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

Kiczales, G. (2002) AOSD 1st. International Conference on Aspect-Oriented Software Development. (Ed.). ACM Press. The Netherlands.

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

Piveta, E. and Zancanela, L. (2003) "Aspect Weaving Strategies", Journal of Universal Computer Science, vol.9, no. 8.

Pryor, J., Diaz Pace, A., Campo, M. (2002) "Reflection on Separation of Concerns". RITA. Vol.9. Num.1.

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

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

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.

Tarr, P., Ossher, H., Harrison W. and Sutton Jr. S.M. (1999) “N Degrees of Separation: Multi-Dimensional Separation of Concerns”. Proceedings of ICSE’99. pp 107-119, IEEE Computer Society Press / ACM Press.

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.