Early Conflicts: Análisis y Resolución de Conflictos Tempranos

Verónica L. Vanoli[1], **Claudia A. Marcos**[2]

[1]Unidad Académica Río Gallegos. Universidad Nacional de la Patagonia Austral
Lisandro de la Torre 1070. CP 9400. Río Gallegos. Santa Cruz. Argentina
Tel/Fax: +54-2966-442313/17.

[2]ISISTAN Research Institute. Facultad de Ciencias Exactas. UNICEN
Paraje Arroyo Seco. CP 7000. Tandil. Buenos Aires. Argentina
Tel/Fax: + 54–2293–440362/3.

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

Resumen. *Actualmente el tratamiento de aspectos en las etapas tempranas de desarrollo de software (aspectos tempranos), es uno de los temas de investigación en la Ingeniería de Software. Durante estas etapas se intenta comenzar a modularizar el sistema identificando, separando y especificando los concerns que cortan transversalmente los componentes funcionales de la aplicación, denominados aspectos candidatos. Entre estos aspectos existen posibles competencias de activación que derivan a situaciones poco deseables llamadas conflictos (conflictos tempranos). El presente trabajo propone un enfoque automatizado para detectar, analizar y resolver toda situación conflictiva que pueda surgir entre aspectos generados tempranamente.*

1. Introducción

La orientación a aspectos busca resolver el problema de separación de concerns [Dijkstra 1976]. El término concern se ha definido como "cualquier materia de interés en un sistema de software" [Sutton 2002], es decir, son diferentes temas o asuntos que son necesario desarrollar para construir un sistema. La separación de concerns disminuye la complejidad que presentan los sistemas y se puede cumplir con requerimientos relacionados con la calidad de diseño, como adaptabilidad, mantenibilidad, extensibilidad y reutilidad. En muchos casos suele suceder que estos concerns aparecen diseminados en el código atravesando partes del sistema no relacionados al modelo, generando lo que se denomina crosscutting concerns [Kiczales 1997]. Es decir, un crosscutting concern es el código disperso y mezclado (scattering y tangling) producto de la implementación de los diferentes concerns de una aplicación en un mismo nivel de código.

El Desarrollo de Software Orientado a Aspectos (DSOA) [Elrad 2001] se ocupa del manejo de crosscutting concerns proveyendo mecanismos para su identificación, separación, representación y composición. Los crosscutting concerns se encapsulan en módulos separados conocidos como aspectos, para promover la localización de los mismos. Resulta así, ser un mejor soporte para la modularización de sistemas reduciendo de esta manera, los costos de desarrollo, mantenimiento y evolución de aplicaciones.

En el DSOA [Kiczales 2002] es frecuente encontrar situaciones en las que un componente de funcionalidad básica se encuentra afectado por dos o más aspectos. Cada uno de los cuales le adiciona diferente comportamiento al componente funcional. Esta situación puede provocar un comportamiento impredecible o indeseado en el sistema, generando así, lo que se conoce como conflicto, interferencia, interacción, etc. [Douence 2002] [Bergmans 2003] [Katz 2004]. Es decir, un conflicto puede ocurrir cuando dos o más aspectos asociados al mismo componente funcional compiten por su activación [Pryor 2002].

La tendencia en el DSOA hacia las etapas tempranas [EA 2004] es relativamente nueva. El área específica de Ingeniería de Requerimientos Orientada a Aspectos (IROA) [Sampaio 2005] intenta proveer soporte para la identificación y separación de las propiedades funcionales y no funcionales. Estas últimas, se denominan aspectos tempranos [EA 2004] o aspectos candidatos [Araújo 2002] y son aquellos que se identifican durante las etapas iniciales del proceso de desarrollo, en especial durante la captura de requerimientos y el diseño arquitectónico [Cuesta 2004]. Otro objetivo de la IROA es proveer mejores medios para la identificación y manejo de conflictos que surjan entre los aspectos tempranos, los que hemos dado en llamar conflictos tempranos (early conflicts), presentado en este trabajo.

El enfoque de este trabajo es detectar, analizar y resolver automáticamente situaciones conflictivas generadas por aspectos identificados en la etapa de ingeniería de requerimientos en el desarrollo del sistema. La funcionalidad del sistema se describe por medio de casos de uso, los cuales brindan información para identificar los aspectos candidatos y detectar las posibles situaciones conflictivas que se generan entre ellos. Para resolver automáticamente dichas situaciones conflictivas, se analizan los casos de uso nuevamente y utilizando técnicas de clasificación (redes bayesianas, árboles de decisión, etc.) se identifica el tipo de conflicto. Una clasificación de conflictos entre

aspectos es propuesta de manera tal de poder clasificarlos e inferir una posible solución. Luego, la solución propuesta se muestra al desarrollador para su aprobación. Durante el análisis los aspectos candidatos ofrecerán información que junto con el conflicto y la resolución aplicada se guardarán en un repositorio; el mismo servirá como base de información de conflictos para futuras aplicaciones.

Este trabajo se organiza de la siguiente manera. En la Sección 2 se profundiza en el concepto de conflictos tempranos. En la Sección 3 se describen los trabajos relacionados a través de un análisis general comparándolos. En la Sección 4 se presenta el enfoque de este trabajo destinado a la detección, análisis y resolución de conflictos entre aspectos, mostrando el proceso propuesto, los niveles de conflictos, la detección, clasificación y resolución de conflictos. Finalmente, en la Sección 5 se exponen las conclusiones.

2. Conflictos Tempranos

En los últimos años, se ha considerado un nuevo tópico como área de investigación y problemática del paradigma a resolver [AOSD 2002], "el fenómeno de los conflictos entre aspectos", también denominados en la literatura como "interacciones" [Douence 2002] [Tanter 2005] o "interferencias" [Bergmans 2003] [Katz 2004]. La ocurrencia de conflictos entre aspectos es una consecuencia de la descomposición de los sistemas de software y su posterior composición en el DSOA. Esto se debe a que una aplicación orientada a aspectos consiste esencialmente de un conjunto de unidades o componentes: clases y aspectos. Es frecuente encontrar situaciones en las que un componente de funcionalidad básica esta afectado por más de un aspecto. Cada uno de los aspectos le adiciona diferente comportamiento provocando una interacción entre éstos. En ciertos casos esta interacción, puede tener por resultado un comportamiento contradictorio, nulo o innecesario. De esta manera, surgen los conflictos entre aspectos y la necesidad de construir mecanismos y estrategias para su adecuado tratamiento. Dicho fenómeno es independiente a las herramientas y requiere una especial atención y tratamiento ya que la activación de ciertos conflictos, pueden provocar comportamientos no deseados e inciertos en la ejecución de los sistemas software.

Cuando dos o más aspectos intenten cortar transversalmente, al mismo tiempo, una determinada funcionalidad básica del sistema y este acceso no afecta al normal desempeño del sistema, es decir, no provoca ninguna anomalía con la ejecución de los aspectos, se dice que la situación conflictiva es una contribución positiva [Chung 2000] y es necesaria detectarla. Caso contrario ocurre si dos o más aspectos alteran el comportamiento del sistema al intentar acceder al mismo tiempo una componente funcional. En este caso, se denomina contribución negativa [Chung 2000] y es necesario, no sólo detectarla sino que resolverla.

Desde el punto de vista de la IROA, definimos a los conflictos tempranos, con el título de early-conflicts, como toda contribución negativa que se genera a partir de que dos o más aspectos tempranos compitan entre sí por su activación.

Realizar un tratamiento temprano de las situaciones conflictivas entre aspectos reduce considerablemente el trabajo que implica identificarlos y resolverlos exclusivamente en la etapa de implementación, es decir, en etapas posteriores o tardías del ciclo de desarrollo de software. Además, se preveen con anterioridad comportamientos ajenos al normal desarrollo del sistema. Y así, lograr la toma de

decisiones con los desarrolladores lo antes posible y no esperar a que se encuentre muy avanzada la construcción del sistema. Es muy importante aclarar que de esta manera se promueve al tratamiento de conflictos para que también sea una tarea a desarrollar en todas las etapas del ciclo de vida del software.

3. Trabajos Relacionados

En la actualidad existen varios proyectos referentes al estudio de aspectos en las etapas tempranas del desarrollo de software, donde se extienden modelos convencionales como viewpoints [Finkelstein 1996] [Sommerville 1997], casos de uso [Jacobson 1992] y orientados a objetivos [Lamsweerde 2001] y proveen un buen soporte para la identificación de la mayoría de los requerimientos del sistema. Algunos de estos proyectos hacen referencia a las situaciones conflictivas que generan los aspectos candidatos, aunque carecen de un tratamiento específico sobre las mismas, es decir, no proponen mecanismos de resolución, ni resuelven.

A continuación se muestran los enfoques analizados teniendo en cuenta los siguientes criterios:

- Niveles de Conflictos: si proveen o no, y en el caso de proveer, qué criterio de nivelación aplican.
- Clasificación y Resolución de Conflictos: la taxonomía de conflictos utilizada.
- Etapa de Resolución: si bien existe una detección temprana, algunos trabajos sugieren la resolución de conflictos para etapas posteriores del desarrollo, por lo tanto, se indica la etapa del desarrollo en la que resuelven los conflictos.
- Tipo de Resolución: manual, semiautomática o automática.
- Observaciones generales: todos aquellos criterios no incluidos en los puntos anteriores, que sean relevantes de los enfoques analizados.

Requerimientos Orientados a Aspectos con UML [Araújo 2002]
No proveen niveles de conflictos. Establecen una clasificación y resolución de conflictos por prioridades (de orden). La etapa de resolución es en la Ingeniería de Requerimientos y lo resuelven de forma manual. Como observaciones: no presentan solución para los casos donde la prioridad es la misma y no proponen solución en los casos donde luego de la resolución, se generen otros conflictos en la composición.

Ingeniería de Requerimientos Orientada a Aspectos [Brito 2004]
No proveen niveles de conflictos. Proponen la siguiente clasificación de resolución: por prioridades (de orden); inhabilitación (de exclusión); sincronización completa (de concurrencia). Resuelven en la etapa de Ingeniería de Requerimientos de forma manual. Observaciones: realizan un uso excesivo respecto a los cambios de prioridades para los concerns; se necesita mucha precisión en los cambios por parte de los desarrolladores para la negociación en la resolución de conflictos y no queda claro si se resuelve la resolución por sincronización completa.

Especificación y Separación de Concerns desde los Requerimientos al Diseño [Kassab 2005]
No proveen niveles de conflictos. Clasificación para resolver: por prioridad (de orden) y proceso de refinamiento de requerimientos. Presentan un framework para la etapa de Ingeniería de Requerimientos pero la resolución es manual. Como observaciones: no utilizan las precondiciones y postcondiciones como provecho para el manejo de conflictos; no es claro el proceso de refinamiento de los requerimientos para la

resolución, ni quiénes participan en él y no presentan solución en los casos que el refinamiento genere otros conflictos.

PROBE [Katz 2004]
Proveen niveles de conflictos según una subdivisión de conflictos, dependiente del framework que presentan. Aunque la resolución es manual y antes de la implementación. Aunque proponen una clasificación según un orden de preferencia y debilidades (de orden) y cambio de requerimientos. Observaciones: se necesita mucha precisión en los cambios por parte de los desarrolladores para la negociación en la resolución de conflictos y la solución por cambio de requerimientos no es clara (no sería recomendable), además, no presentan solución en los casos en que dichos cambios generen otros conflictos.

Detección de Conflictos entre Crosscutting Concerns, basado en el Modelado [Tessier 2004]
Proveen niveles de conflictos según una categorización de los conflictos y dos niveles diferentes para resolverlos. No establecen clasificación y resolución de conflictos alguna. Utilizan una herramienta, pero la resolución es manual. Como observaciones: de los dos niveles propuestos, sólo trabajan con uno de ellos, dejando de lado los conflictos más difíciles de detectar.

Aspects Extractor [Haak 2006]
No proveen niveles de conflictos ni clasificación para resolver los conflictos. La etapa de tratamiento de conflictos es en la de Ingeniería de Requerimientos y para ello utilizan una herramienta, pero se resuelve manualmente. Observaciones: sólo detectan las situaciones conflictivas y el desarrollador se encarga de de tratar los conflictos.

Teniendo los enfoques, existen dos puntos principales a tener en cuenta que son comunes a los enfoques descriptos.

En primer lugar, aunque algunos de estos enfoques presentan herramientas para identificar aspectos tempranamente, la tarea que concierne al tratamiento de conflictos es totalmente manual. Esto hace que el desarrollador se haga cargo absolutamente de la toma de decisiones, no existiendo así un mecanismo automático o semiautomático, que ayude a resolver los conflictos sin la intervención de los desarrolladores.

En segundo lugar, es la no existencia de una amplia clasificación para la resolución de conflictos. No se ofrecen alternativas de resolución. En su mayoría, se proponen niveles de prioridades para que los conflictos sean resueltos en un orden específico. Y para destacar, muchos de estos enfoques establecen información interesante en la descripción de los requerimientos, no aprovechada como material en la resolución de conflictos.

4. Enfoque para Analizar y Resolver Conflictos tempranos

En este trabajo se propone automatizar la detección y resolución de conflictos entre aspectos tempranos. Para ello se toma como base la Herramienta *Aspects Extractor* [Haak 2006] y se le adiciona la posibilidad de resolver las situaciones conflictivas entre los aspectos candidatos que la herramienta identifica.

Esta herramienta está conformada por una serie de tareas automatizadas, donde se analiza la información provista por el usuario y se extraen datos de interés que llevarán a la obtención de los aspectos candidatos. Finalmente, se necesita la decisión

del desarrollador, quien seleccionará los aspectos definitivos. La funcionalidad del sistema se describe por medio de casos de uso y se identifican automáticamente los aspectos candidatos de la aplicación. Las cinco tareas principales automatizadas que forman parte del modelo de ingeniería de requerimientos propuesto por este enfoque son: Identificar concerns, Seleccionar aspectos candidatos, Especificar aspectos candidatos, Identificar conflictos y Modelar en UML.

La Herramienta *Aspects Extractor*, como antes se ha mencionado, en su modelo de ingeniería de requerimientos, define la Tarea 4 titulada Identificar conflictos. En este trabajo se aplica una extensión a dicha tarea para que esos conflictos puedan ser analizados para su resolución posterior. La Figura 1 describe el conjunto de tareas propuesto para identificar (detectar), analizar y resolver las posibles situaciones conflictivas.

Figura 1. Extensión del Modelo de Ingeniería de Requerimientos de la Herramienta *Aspects Extractor*.

Dicha tarea ahora se denomina Administrar conflictos y consta de las siguientes tres Subtareas: 4.1. Detectar conflictos entre aspectos candidatos, en donde se realiza la detección de los conflictos que se generan entre aspectos identificados; 4.2. Analizar información del sistema, en este caso se analiza toda la información del sistema necesaria para resolver los conflictos, principalmente extraída de los casos de uso; y 4.3. Resolver conflictos entre aspectos, donde se aplica algún tipo de resolución a los conflictos detectados.

En la próxima sección, el proceso de identificación, análisis y resolución de conflictos se describen con más detalle para su mejor comprensión.

4.1. Proceso Propuesto

La Figura 2 muestra en detalle el proceso propuesto en este trabajo para administrar tempranamente los conflictos. El proceso constituye el eje principal del enfoque automatizado.

El proceso contiene un conjunto de actividades a realizar y un componente repositorio, que se detallan a continuación:

- **Detectar Conflictos entre Aspectos Candidatos**: la Herramienta *Aspects Extractor* identifica las posibles situaciones conflictivas entre aspectos que afectan a un mismo caso de uso, generando así un listado de las mismas que formarán parte del proceso de administración para ser resueltas.
- **Buscar en Repositorio Conflictos Evidentes**: es la tarea que se encarga de determinar la existencia o no de conflictos evidentes ya recopilados en un repositorio. Es decir, aquellas situaciones conflictivas resueltas con éxito en aplicaciones anteriores. Para esta actividad se busca en el repositorio de conflictos evidentes, el cual es la base de información donde se almacenan las resoluciones de conflictos exitosas. El repositorio contiene los aspectos candidatos con sus

respectivas situaciones conflictivas y la resolución adoptada con anterioridad. Dicho repositorio, se actualizará automáticamente en el transcurso del tiempo con su utilización, pero también puede existir una intervención manual ya sea con anterioridad o durante su uso. La información principal del repositorio se encuentra conformada por: los aspectos candidatos, el conflicto generado e identificado entre ellos, el nivel de conflicto al que pertenece y la resolución aplicada en tal caso.

- **Analizar Casos de Uso**: todo aquel dato que surja de la especificación de los casos de uso, servirá como información en esta actividad para tomar decisiones de resolución a todo conflicto que se genere. Las decisiones se basan en realizar un análisis detallado de la especificación de los casos de uso tratando de inferir la información necesaria que ayude a identificar el tipo de conflicto y sus posibles soluciones.
- **Interactuar con Desarrolladores**: cuando no exista posibilidad alguna para resolver conflictos en forma automatizada o existan varias alternativas de resolución, los desarrolladores podrán aportar información necesaria para la resolución.
- **Resolver Conflictos entre Aspectos Candidatos**: es donde se ofrece una solución a las situaciones conflictivas que será aprobada o no por el desarrollador.

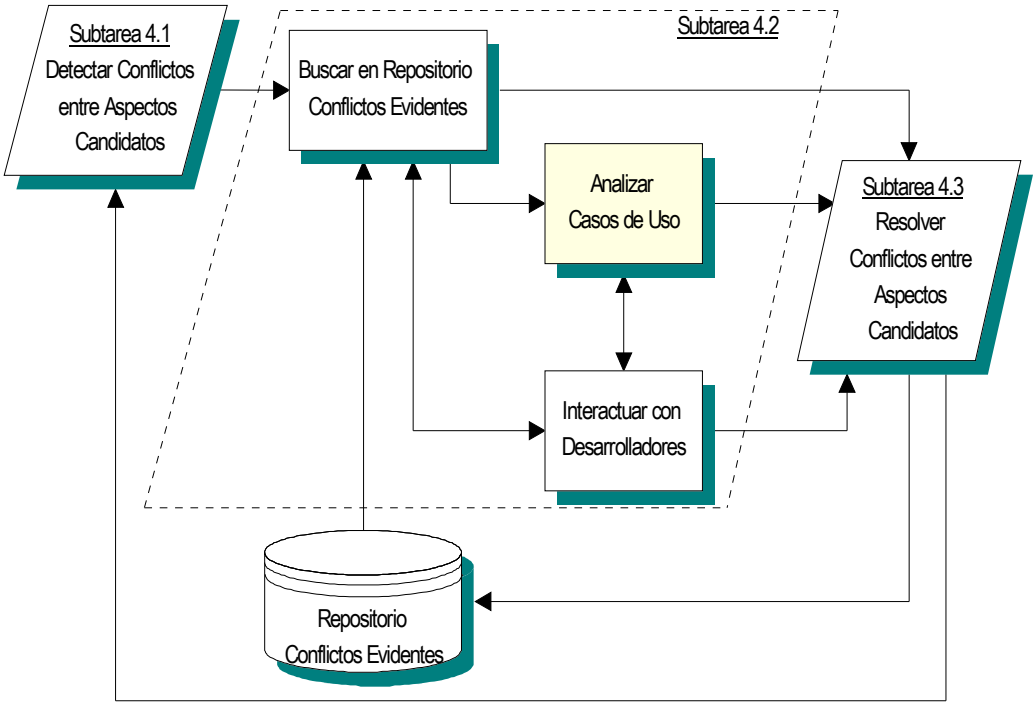

Figura 2. Proceso de Administración Temprana de Conflictos entre Aspectos.

El flujo de actividades del proceso de administración temprana de conflictos tiene como entrada una lista de conflictos generada a partir de los aspectos candidatos identificados por la Herramienta *Aspects Extractor* (Subtarea 4.1). Por ejemplo, en un sistema de alumnos se pueden registrar los aspectos candidatos: logeado, autorizado y persistencia. Estos tres casos, se encuentran afectados a los casos de uso: dar de alta a un alumno, inscribir en una materia, cancelar inscripción a una materia, inscribir para

rendir un final, entre otros. En consecuencia, existe una situación conflictiva entre los aspectos.

La zona marcada con líneas punteadas representa a la Subtarea 4.2, donde se analiza la información para la toma de decisiones respecto a cómo se van a tratar los conflictos, para su posterior resolución. En dicho análisis, se realiza en primer término la búsqueda de conflictos evidentes en el repositorio de conflictos evidentes. Si la búsqueda resulta exitosa, automáticamente se presenta al desarrollador, para su aprobación, la resolución de la situación conflictiva. Caso contrario, es decir, la búsqueda fracasa, se procede a analizar la descripción y propiedades de los casos de uso, por los cuales los aspectos candidatos en conflicto compiten por su activación.

Respecto a los casos de uso, los ítems correspondientes a la especificación de los mismos (descripción, actores, triggers, suposiciones, precondiciones, flujos básicos, flujos alternativos, postcondiciones, terminación, requerimientos especiales, extensión, inclusión, generalización, frecuencia, temas abiertos, lista de tecnología, etc.) resultan ser una fuente sumamente útil para efectuar el análisis. Como resultado del mismo puede ocurrir que el caso de uso no proporcione o no contenga información suficiente como para una correcta toma de decisión, en este caso, se hace partícipe al desarrollador que corresponda, permitiendo una interactividad donde la toma de decisión queda a cargo del mismo. Caso contrario, si el análisis resulta favorable, es decir, se puede rescatar información del caso de uso, se procede a la resolución del conflicto. El análisis se desarrollará también, teniendo en cuenta las técnicas de clasificación, por medio de los cuales el sistema tendrá un conocimiento de experiencias pasadas que ayudará a la identificación de las soluciones a los conflictos de la aplicación en desarrollo.

La Subtarea 4.3, es la resolución de los conflictos identificados. Para ello se toma en cuenta la clasificación y resolución de conflictos. El resultado de la resolución vuelve a ser analizado para verificar que no existan nuevas situaciones conflictivas generadas por la resolución adoptada. En caso que la resolución propuesta genere nuevos conflictos nuevamente se iniciará el proceso de resolución de conflictos. Este proceso se aplicará hasta resolver todos los posibles conflictos o hasta que el desarrollador decida dejar su resolución para etapas futuras del desarrollo. Cuando la solución propuesta sea la deseada y no existan más conflictos se actualizará en el repositorio los datos referentes a cada conflicto detectado y resuelto y sus respectivos aspectos.
Por otra parte, existirá un diagrama de casos de uso extendido modelado en UML, con nuevos estereotipos que describan visualmente los aspectos candidatos y las situaciones conflictivas con la resolución aplicada. Es decir, se definirá un perfil UML para la ingeniería de requerimientos aplicado a aspectos. De esta manera quedan registrados los aspectos para proseguir con el desarrollo del sistema.

4.2. Niveles de Conflictos

Basándose en las dos clases de contribuciones (positiva y negativa) y para un mejor ordenamiento de los conflictos previo a la resolución y agilizando la misma, hemos establecido cuatro niveles de situaciones conflictivas.

La descripción de los cuatro niveles se expresará en un lenguaje natural (explicando claramente de qué se trata cada uno) y en un lenguaje formal (como

representación simbólica de cada uno). Para este último caso, previamente, es necesario definir el siguiente conjunto:

A = {Aspecto Candidato}
CU = {Caso de Uso}
C = {Conflicto}
R = {Repositorio de Conflictos)
Un conflicto se define como:

$$\exists\ A_1 / A_1 \in CU_x \wedge \exists\ A_2 / A_2 \in CU_x \wedge \exists\ A_3 / A_3 \in CU_x \wedge ... \wedge \exists\ A_n / A_n \in CU_x \Leftrightarrow A_1, A_2, A_3, ..., A_n \subseteq C_x$$

- **Interferencia libre**: es toda aquella competencia de activación entre aspectos candidatos que no genera ningún tipo de conflicto. Es decir, es la ausencia de conflictos. Si bien los aspectos candidatos compiten por su activación, no se afectan unos a los otros. Este sería el caso de una contribución positiva y no se debe tomar ninguna medida de resolución.

$$A_1, A_2, A_3, ..., A_n \not\subset C_x$$

- **Conflicto evidente**: es toda aquella situación conflictiva conocida, es decir, ya ha sido detectada con anterioridad y presenta una posible solución. Este es el caso donde se podría carecer del análisis de la información, dado que la resolución ya fue aplicada, sólo bastaría con elegirla y proseguir. A menos que el desarrollador quiera optar por otra o por intermedio de las técnicas de clasificación se proporcione una mejor alternativa de resolución y de esta forma, se mantiene la automatización. Con el tiempo, este será uno de los niveles más aplicados.

$$A_1, A_2, A_3, ..., A_n \subseteq C_x \wedge C_x \in R$$

- **Conflicto avanzado**: es toda aquella competencia de activación entre nuevos aspectos candidatos identificados que no han sido encontrados en aplicaciones anteriores, es decir que no son conflictos evidentes. Esta situación se va a presentar en los comienzos de este enfoque y a medida que la tecnología imponga nuevos aspectos en los sistemas de software. Este es el caso donde la resolución de conflictos participa en su totalidad.

$$A_1, A_2, A_3, ..., A_n \subseteq C_x$$

- **Conflicto intermedio**: es toda aquella competencia de activación entre aspectos candidatos identificados que no son conflictos evidentes y aquellos que sí lo son, es decir una combinación de los que han estado involucrados anteriormente en situaciones conflictivas y aquellos que no. Este es el caso donde se deberá resolver la situación conflictiva, pero teniendo en cuenta que alguno de los aspectos candidatos involucrados ya han participado de otro conflicto que fue registrado; este nivel de conflicto es el que aumenta las posibilidades de nuevos conflictos, por lo tanto, es aquí donde se debería tener absolutamente en cuenta volver a analizar el resultado de la resolución para verificar que no existan nuevas situaciones conflictivas generadas por la resolución adoptada.

$$A_1, A_2, A_3, ..., A_n \subseteq C_x \wedge (A_1 \vee A_2 \vee A_3) \in R$$

4.3. Detección de Conflictos

La herramienta *Aspects Extractor* analiza la información de los casos de uso y si un caso de uso se encuentra afectado por más de un aspecto candidato, entonces existe una situación conflictiva, por lo tanto, se la identifica. De esta manera todas las posibles situaciones conflictivas son identificadas.

4.4. Clasificación y Resolución de Conflictos

La clasificación que se propone para la resolución de conflictos en la etapa temprana del desarrollo de software, se encuentra ampliada y adaptada de acuerdo a la taxonomía de resolución de [Pryor 2003]. Hasta el momento se ha identificado una clasificación de nueve tipos de políticas de resolución en caso de activación de conflictos: en orden, en orden inverso, en orden natural, por partición, por unión, por inclusión compuesta, por inclusión simple, combinaciones y por nulidad.

Al momento de resolver las situaciones conflictivas generadas por los aspectos candidatos, el orden de aplicación de los distintos tipos de resolución se encontrará ligada a la cantidad de aspectos conflictivos que se van a ejecutar (todos, algunos o ninguno) según el tipo de conflicto. El mejor caso de ellos se logra cuando se conserva la integridad de los aspectos, es decir, cuando los aspectos en conflicto puedan ser ejecutados al solucionar la situación que los lleva al mismo.

A continuación se presenta la clasificación de conflictos propuesta para la resolución, según el orden de aplicación (de igual forma que los niveles de conflictos, expresada en lenguaje natural y formal):

Todos los aspectos

- **En Orden**. Se establece un orden para los aspectos candidatos en conflicto, planteando prioridades para su activación.

$$C_x \rightarrow O(A_1, A_2, A_3, ..., A_n) \Rightarrow A_i, A_j, A_k, ..., A_z \; \forall \; i=[1 \mid 2 \mid 3 \mid ... \mid n] \wedge \forall \; j=[1 \mid 2 \mid 3 \mid ... \mid n] \wedge \forall \; k=[1 \mid 2 \mid 3 \mid ... \mid n] \wedge \forall \; z=[1 \mid 2 \mid 3 \mid ... \mid n] \wedge i \neq j \neq k \neq z$$

- **En Orden Inverso**. Se establece un orden invertido para los aspectos candidatos en conflicto. De la misma manera que el caso anterior, sólo que las prioridades planteadas se invierten. Este caso se aplica cuando intervenga el desarrollador.

$$C_x \rightarrow OI(A_1, A_2, A_3, ..., A_n) \Rightarrow A_i, A_j, A_k, ..., A_z \; \forall \; i=[1 \mid 2 \mid 3 \mid ... \mid n] \wedge \forall \; j=[1 \mid 2 \mid 3 \mid ... \mid n] \wedge \forall \; k=[1 \mid 2 \mid 3 \mid ... \mid n] \wedge \forall \; z=[1 \mid 2 \mid 3 \mid ... \mid n] \wedge i \neq j \neq k \neq z \Rightarrow A_z, ..., A_k, A_j, A_i$$

- **Por Partición**. Se determina que un aspecto candidato en conflicto pueda ser divido al menos en dos partes, generando así la posibilidad de crear nuevos aspectos candidatos que no generen situaciones conflictivas.

$$C_x \rightarrow P(A_1, A_2, A_3, ..., A_n) \Rightarrow A_{1.1}... A_{1.m}, A_{2.1} ... A_{2.m}, A_{3.1}... A_{3.m}, A_{n.1}... A_{nm}$$

- **Por Unión**. Se determina que dos o más aspectos candidatos en conflicto puedan unificarse al menos en un solo aspecto candidato y así resolver la o las situaciones conflictivas.

$$C_x \rightarrow U(A_1, A_2, A_3, ..., A_n) \Rightarrow A_1 \cup A_2 \cup A_3 \cup ... \cup A_n = A_u \Rightarrow A_u$$

Alguno de los aspectos

- **Por Inclusión Compuesta**. Se determina que alguno de los aspectos candidatos en conflicto quedará incluido en el modelo, por lo tanto, el resto de los aspectos candidatos serán excluidos. Luego de determinar los aspectos incluidos, se vuelve al inicio de la taxonomía para determinar como se ejecutarán los mismos.

$$C_x \rightarrow IC(A_1, A_2, A_3, ..., A_n) \Rightarrow A_1, A_2, A_3, ... A_m \; \forall \; m<n$$

- **Por Inclusión Simple**. Se determina que sólo un aspecto candidato en conflicto quedará incluido en el modelo, por lo tanto, el resto de los aspectos candidatos serán excluidos.

$$C_x \rightarrow IS(A_1, A_2, A_3, ..., A_n) \Rightarrow A_i \; \forall \; i=[1 \mid 2 \mid 3 \mid ... \mid n]$$

Ninguno de los aspectos

- **Por Nulidad**. Se determina la nulidad total de los aspectos candidatos en conflicto. De esta manera, todos los aspectos candidatos quedarán excluidos del modelo.

$$C_x \rightarrow N(A_1, A_2, A_3, ..., A_n) \Rightarrow \neg(A_1 \vee A_2 \vee A_3 \vee ... \vee A_n) \Rightarrow \varnothing$$

Existen también otras posibilidades cuando las políticas anteriores no se puedan aplicar o no sean suficientes para la resolución de conflictos, como lo son:

- **En Orden Natural**. Se establece un orden según como la Herramienta Aspects Extractor devuelve listado a los aspectos candidatos en conflicto.

$$C_x \rightarrow ON(A_1, A_2, A_3, ..., A_n) \Rightarrow A_1, A_2, A_3, ..., A_n$$

- **Combinaciones**. Se puede realizar combinaciones entre políticas y así lograr el objetivo deseado. Este caso se aplica al desarrollador, es decir, cuando lo desee puede optar por elegir más de una de las políticas presentadas para resolver los conflictos.

Esta clasificación se tendrá en cuenta según el nivel de conflicto establecido previamente. Logrando que los niveles sean modificables, ya que pueden cambiar posteriormente de acuerdo a la política de resolución elegida y aplicada.

Al identificar el tipo de conflicto automáticamente se realizará un análisis de la especificación de los casos de uso involucrados en la situación conflictiva. Por ejemplo, para aplicar un orden, se puede tener en cuenta los siguientes ítems de las especificaciones: la prioridad de los aspectos (en el ejemplo visto anteriormente del sistema de alumnos, el aspecto candidato logeado tiene una prioridad más alta que persistencia, por lo tanto, el orden se daría en primer lugar con el logeo y a continuación con persistencia); trigger, precondiciones, postcondiciones y terminación del caso de uso (teniendo en cuenta el orden en que aparecen los aspectos en cada uno de los ítems); el flujo básico del caso de uso (en el ejemplo del sistema de alumnos, los alumnos deberán estar primero autorizados a rendir un final antes que poder registrar ese final); el flujo alternativo del caso de uso; etc. Para aplicar partición, se puede tener en cuenta el ítem generalización del caso de uso o del aspecto. En el caso de unión, los ítems extensión e inclusión del caso de uso o del aspecto. Los ítems de requerimientos especiales o no-funcionales y de suposiciones del caso de uso, se pueden aplicar para un caso de inclusión compuesta o simple (en el sistema de alumnos, si la suposición es que el alumno deba estar logeado, esto permite que se incluya solamente dicho aspecto

candidato). Entre otros casos. Si la especificación de los casos de uso y los aspectos no es suficiente se podrá adicionar otro dato del caso de uso que pueda aportar información, lo mismo para los aspectos candidatos.

5. Conclusiones

Una de las contribuciones más importantes de este trabajo es el proceso automático para la administración (identificación, análisis y resolución) de conflictos en las primeras etapas del desarrollo de aplicaciones orientadas a aspectos.

Los niveles de conflictos en colaboración con el repositorio de conflictos ayudan a organizarlos para lograr clasificarlos y aplicar una correcta resolución.

Se ha definido un conjunto de tipos de conflictos con sus correspondientes resoluciones. La identificación y resolución esta basada en un análisis semántico y sintáctico de las descripciones de los casos de uso por medio de los cuales será posible inferir el tipo de conflicto con una posible solución. Esto permite resolver las diferentes situaciones conflictivas de manera precisa y automática.

La utilización de técnicas de clasificación es un mecanismo novedoso para la clasificación y resolución de los conflictos entre aspectos de forma automática.

El seguimiento, tanto de los conflictos resueltos como de los identificados pero no resueltos, ayuda en el desarrollo, documentación y mantenimiento de las aplicaciones orientadas a aspectos. En caso que surja algún cambio en las etapas posteriores o una vez que el sistema ha sido terminado, será posible obtener la información de los aspectos involucrados en esos conflictos y modificarlos sin necesidad de buscar en todo el sistema.

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

Referencias

[AOSD 2002] Homepage Aspect-Oriented Software Development (AOSD): http://www.aosd.net. 2002.

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

[Bergmans 2003] Bergmans L. M. J. “Towards detection of semantic conflicts between crosscutting concerns”. In ECOOP: AAOS ’03: The first workshop on Analysis of Aspect-Oriented Software, Darmstadt, Germany, July 21 2003.

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

[Chung 2000] Chung L., Nixon B., Yu E., Mylopoulos J. “Non-Functional Requirements in Software Engineering”. Kluwer Academic Publishing. 2000.

[Cuesta 2004] Cuesta C., Romay P., de la Fuente P., Solórzano M. “Aspectos como Conectores en Arquitectura de Software”. II Jornadas DYNAMICA. España. Noviembre de 2004.

[Dijkstra 1976] Dijkstra E.W. “A Discipline of Programming”. Prentice-Hall. 1976.

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

[EA 2004] Homepage Early Aspects (EA): Aspect-Oriented Requirements Engineering and Architecture Design. http://www.early-aspects.net/. 2004.

[Elrad 2001] Elrad T., Filman R., Bader A. “Theme Section on Aspect-Oriented Programming”. Communications of ACM, Vol. 44 No. 10. 2001.

[Finkelstein 1996] Finkelstein A., Sommerville I. “The Viewpoints FAQ”. BCS/IEE Software Engineering Journal, vol. 11, 1.996.

[Haak 2006] Haak B., Díaz M., Marcos C., Pryor J. “Aspects Extractor: Identificación de Aspectos en la Ingeniería de Requerimientos”. IDEAS’06 9° Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software. Facultad de Informática, Universidad Nacional de La Plata. 24 a 28 de Abril de 2006.

[Jacobson 1992] Jacobson I. “Object-Oriented Software Engineering: A Use Case Driven Approach”, 4 ed: Addison-Wesley, 1.992.

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

[Katz 2004] Katz A., Rashid A. “PROBE: From Requirements and Design to Proof Obligations for Aspect-Oriented Systems”. Computing Department Lancaster University Technical Report Number: COMP-002-2004. February 2004.

[Kiczales 1997] Kiczales G., Lamping J., Menhdhekar A., Maeda C., Lopes C., Loingtier J-M., Irwin J. “Aspect-Oriented Programming”. In Proceedings European Conference on Object-Oriented Programming (ECOOP), LNCS 1241, pages 220–242. Finland. Springer-Verlag. June 1997.

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

[Lamsweerde 2001] Lamsweerde A. “Goal-Oriented Requirements Engineering: A Guided Tour”. Presented at 5th IEEE International Symposium on Requirements Engineering. 2001.

[Pryor 2002] Pryor J., Diaz Pace A., Campo M. “Reflection on Separation of Concerns”. RITA. Vol 10, Num 2. 2002.

[Pryor 2003] Pryor J., Marcos C. “Solving Conflicts in Aspect-Oriented Applications”. Proceeding of the Fourth ASSE. 32 JAIIO. Universidad Argentina de la Empresa. 1 al 5 de Septiembre de 2003.

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

[Sommerville 1997] Sommerville I, Sawyer P. “Viewpoints: Principles, Problems and a Practical Approach to Requirements Engineering”. Annals of Software Engineering, vol. 3, pp. 101-130, 1.997.

[Sutton 2002] Sutton S. M. Jr., Rouvellou I. “Modelling of Software Concerns in Cosmos”. En Proceedings of the 1st International Conference on Aspect-Oriented Software Development. ACM Press. 2002.

[Tanter 2005] Tanter E., Noye J. “A Versatile Kernel for Multi-Language AOP”, Proceeding of ACM SIGPLAN/SIGSOFT – Conference on Generative Programming and Component Engineering (GPCE’05) LNCS, Springer-Verlag, Estonia, 2005.

[Tessier 2004] Tessier F., Badri M., Badri L. “A Model-Based Detection of Conflicts Between Crosscutting Concerns: Towards a Formal Approach”. International Workshop on Aspect-Oriented Software Development (WAOSD). Peking University, Beijing, China. September 2004.