Contratos traslúcidos: Implementación de Diseño por Contrato en el Desarrollo Orientado a Aspectos

Guadalupe Isaura Trujillo-Tzanahua y Ulises Juárez-Martínez

División de Estudios de Posgrado e Investigación,
Instituto Tecnológico de Orizaba
Orizaba, Ver., México
gittz87@hotmail.com, ujuarez@ito-depi.edu.mx
Paper received on 24/07/12, Accepted on 10/09/12.

Resumen. Un contrato traslúcido es el resultado de combinar 1) la Programación Orientada a Aspectos (POA) la cual ofrece ventajas como: separación de asuntos, facilidad para razonar conceptos, reutilización, mejor mantenimiento del software, entre otras; 2) implementar el Diseño por Contrato (DbC) de forma dinámica, es decir que conforme al evento que identifique una regla pueda aplicar o no. En este artículo se presentan los elementos propuestos en el Enfoque de Temas y los resultados obtenidos en el desarrollo de software para una aplicación de Domótica combinando la POA y los contratos traslúcidos en Ptolemy.

Palabras Clave: Contratos Traslúcidos, Diseño por Contrato, POA, Ptolemy**.**

1 Introducción

El objetivo de la Ingeniería de Software es mejorar la calidad del software mediante la reducción de costos de producción, facilitar su mantenimiento y evolución. Para lograr este objetivo han surgido diversos paradigmas de programación que buscan reducir la complejidad del software y promueven la reutilización y desarrollo de software fiable. Dentro de los paradigmas de programación se encuentra la Programación Orientada a Objetos (POO) la cuál busca el desarrollo de software de calidad a través de la abstracción, encapsulación, polimorfismo y descomposición, no obstante presenta limitaciones que dificultan el mantenimiento y evolución del software. La descomposición aplicada en la POO consiste en dividir el sistema en varios subsistemas que corresponden a diferentes partes del dominio, sin embargo no se adapta a algunos requisitos no funcionales tales como: manejo de excepciones, seguridad, sincronización, registro (*logging*), persistencia, entre otros. Con el advenimiento de la Programación Orientada a Aspectos (POA) que cada vez es más utilizada en áreas como Aviación, Domótica, Redes de telecomunicaciones, por

Advances in Intelligent and Information Technologies.
Research in Computing Science 60, 2012, pp.165 -174.

mencionar algunas, debido a que permite una adecuada modularización de las aplicaciones encapsulando los asuntos de corte, es decir asuntos que interfieren con la funcionalidad de otros. Tras observar que el Diseño por Contratos (DbC) es una técnica muy conocida pero poco utilizada en el desarrollo de software, se infiere la utilidad de combinar estos conceptos (DbC y POA) para aprovechar al máximo las ventajas que cada uno ofrece. La combinación del DbC y el enfoque orientado a aspectos da origen a un mecanismo más robusto denominado contrato traslúcido cuya implementación es provista por Ptolemy, un lenguaje orientado a aspectos con soporte completo para la programación basada en eventos. Sin embargo, no se reporta una guía para el desarrollo de software orientado a aspectos utilizando contratos traslúcidos.

La contribución principal de este trabajo consiste en especificar mediante un caso de estudio aplicado a la Domótica, cómo construir correctamente software en Ptolemy aplicando este concepto desde las etapas de análisis y diseño del desarrollo de software orientado a aspectos (DSOA) utilizando el Enfoque de Temas.

Este artículo está organizado de la siguiente forma: en la sección 2 se describe el lenguaje de programación orientado a aspectos llamado Ptolemy. La sección 3 se refiere a la técnica del Diseño por Contratos. La sección 4 describe a los contratos traslúcidos. La sección 5 presenta el Enfoque de Temas. La sección 6 muestra los resultados obtenidos mediante un caso estudio. La sección 7 presenta los trabajos relacionados. Finalmente, en la sección 8 se presentan las conclusiones y el trabajo a futuro.

2 Ptolemy

Ptolemy es un lenguaje de programación cuyo objetivo es mejorar la capacidad de un ingeniero para separar los asuntos conceptuales. Asimismo, añade una nueva noción de contratos flexibles denominados contratos translúcidos [1].

Cuando se implementan algunos requisitos utilizando las técnicas orientadas a objetos por ejemplo: manejo de excepciones, sincronización, compartir recursos, *logging*, por mencionar algunos, el código de esos requisitos se dispersa en todo el software y se confunde con el código de otros requisitos. Estos requisitos se denominan **asuntos de corte (*crosscutting concerns*)**. Cuando los desarrolladores atienden las solicitudes de mantenimiento a este tipo de requisitos es necesario que examinen un gran número de módulos del sistema para identificar los cambios que se requieran.

El objetivo de Ptolemy es establecer mecanismos que permitan a los desarrolladores de software implementar los asuntos de corte en módulos separados llamados ***handlers***, los cuales mejoran la separación de asuntos [2].

Ptolemy extiende a Java con nuevos mecanismos para la declaración de eventos y avisos a estos eventos. Asimismo permite a los desarrolladores escribir contratos traslúcidos como parte de la definición del tipo de evento.

Ptolemy se distingue de lenguajes OO como Java debido que ofrece el soporte para el manejo de eventos y de los Lenguajes Orientados a Aspectos como AspectJ y CaesarJ porque solventa dificultades identificadas tales como:

1. **Cuantificación (quantification**): referenciar los lugares en el código donde un evento es disparado utilizando el identificador del evento. Esto se hace sin tener que nombrar a las clases que disparan el evento.
2. **Inconsciencia (*obliviousness*):** liberar al programador de las especificaciones puntuales donde una sentencia o conjunto de sentencias del programa tendrá efecto.
3. **Fragilidad en los cortes:** dependencia de los cortes respecto a cómo se estructura el código sobre el que actuarán los aspectos**.**
4. **Sombras en los puntos de unión (Join point shadows**): instrucción en el bytecode donde los avisos se pueden aplicar.
5. **Control de avisos**: los avisos deben ser comprendidos con el fin de razonar el flujo de un programa y cómo un aviso puede interferir con la ejecución de otro aviso [3].
6. **Razonamiento modular:** la formalización del Diseño por Contrato en la POA dificulta el razonamiento de los efectos de un aspecto sobre otros [4], es decir se debe considerar:
   - Todos los posibles puntos de unión en el código base.
   - Todos los aspectos aplicables en cada punto de unión.
   - Por cada aspecto, el control de flujo de los aspectos deben ser comprendidos, es decir el desarrollador es capaz de conocer todo lo que ocurre en un módulo funcionalmente independiente (alta cohesión y bajo acoplamiento).

3 Diseño por Contrato (DbC)

El Diseño por Contrato (DbC) es una técnica diseñada por Bertrand Meyer y característica central del lenguaje de programación Eiffel, que él desarrolló [5]. El DbC no es específico para Eiffel, puede ser utilizado con cualquier lenguaje de programación para desarrollar software correcto, es decir que se comporte de acuerdo a su especificación.

Un contrato es una especificación que establece las condiciones de uso e implementación (obligaciones y beneficios) que clientes y proveedores de un componente deben cumplir. Consecuentemente, un contrato protege al cliente por especificar lo que necesita y por el otro al proveedor por especificar los servicios que debe implementar.

Una regla importante para el DbC es el principio de no cláusulas ocultas [5], es decir, la precondición es el único requisito que un cliente debe cumplir para obtener un servicio, por lo tanto no hay sorpresas.

Una aseveración es una condición que debe cumplirse. Su incumplimiento invalida totalmente el software. En el DbC se utilizan tres tipos de aseveraciones:

1. **Precondición:** expresa las restricciones bajo las cuales una rutina funcionará correctamente.
2. **Postcondición:** describe el efecto de la rutina.
3. **Invariante de clase:** estado que deben cumplir los objetos de una clase, son restricciones que deben satisfacerse por cada objeto tras la ejecución de los métodos y constructores.

4 Contratos traslúcidos

Un contrato traslúcido para un tipo de evento es un algoritmo abstracto que describe el comportamiento de los aspectos que se aplican a una interface orientada a aspectos. El algoritmo es abstracto porque suprime muchos detalles de la implementación. Esto permite a la especificación decidir qué detalles ocultar y cuáles revelar. Un fragmento de código satisface la especificación de un algoritmo abstracto si el código refina la especificación, pero usa de forma limitada el refinamiento que requiere de una estructura similar, lo cual permite el control de efectos [2]. Un contrato traslúcido se puede expresar como:

DbC + POA= Contratos traslúcidos

En [6] se propone a C# para la aplicación de contratos traslúcidos debido que son independientes de su contexto original (Ptolemy) y son aplicables a otras interfaces orientadas a aspectos como Crosscut Programming Interfaces (XPIs), Open Modules (OMs) y Aspect-Aware Interfaces (AAIs).

Los compromisos que deben cumplir los contratos traslúcidos son [7]:

1. Los contratos traslúcidos deben revelar todas las llamadas a **invoke.**
2. Todos los *handlers* deben refinar el contrato de un evento.

4.1 Definición de un contrato traslúcido

La sintaxis para la definición de contratos traslúcidos se muestra a continuación:

01 < Evento>:= [Modificador *] [Tipo] **event** [Identificador] {[Contrato]}
02 <Contrato>:= **requires** <Expresión> **assumes** <Sentencia> **ensures** <Expresión>

En la figura 1 se muestra la implementación del contrato traslúcido para el caso de estudio de la aplicación de Domótica que se encarga de monitorizar cuando algún elemento de la casa (climatización, iluminación o alarma) es encendido o apagado.

```
01  public void event EventoCasa {
02     DomoticaElement eventoCasa;
03    //Contrato traslúcido para este tipo de evento
04    requires eventoCasa != null //precondición
05    assumes {  //invariante de clase
06    next.invoke();
07    establishes next.eventoCasa() == old(next.eventoCasa());
08    }
09    ensures eventoCasa != null //postcondición
10  }
```

Figura 1. Implementación de un contrato traslúcido para la aplicación Domótica.

5. Enfoque de Temas

El Enfoque de Temas permite identificar y modelar aspectos independientemente del lenguaje orientado a aspectos a utilizar. Clarke y Baniassad establecen que el enfoque de temas se compone de dos partes [8]:

1. **Theme/Doc (Análisis):** conjunto de heurísticas para el análisis de los requisitos de software.
2. **Theme/UML (Diseño):** permite al desarrollador modelar en UML temas (características y aspectos) de un sistema.

Un **tema** [9] es un elemento de diseño, es decir una colección de estructuras y comportamientos que representan una característica. Varios temas pueden integrarse o combinarse para formar un sistema. El modelo de temas considera dos tipos de temas [10]:

1. **Temas Base**: los cuales comparten algunas estructuras y comportamiento con otros temas base.
2. **Temas de Corte:** aspectos, es decir temas que poseen comportamiento que corta transversalmente la funcionalidad de los temas base.

5.1 Theme/Doc

El proceso de análisis Theme/Doc tiene dos actividades principales:

1. Identificar los temas principales en un sistema
2. Determinar si las responsabilidades de cierto tema deben ser modelados como un aspecto.

Una vista es un diagrama que engloba requisitos. Theme/Doc se compone de 3 principales vistas [8]:

1. **Vistas de relaciones**: muestran las relaciones entre los temas.
2. **Vistas de corte:** los requisitos comunes entre temas describen el comportamiento que corta transversalmente a otros temas (corte). Es decir los requisitos se encuentran dispersos en varios temas (asuntos de corte) del software. Sin embargo, si estos temas de corte se identifican correctamente pero no se manejan adecuadamente se presentan las dificultades anteriormente citadas (sección 2). Para solventar estos problemas desde las fases de análisis y diseño, se consideró utilizar los contratos traslúcidos, por lo cual es necesario definir extensiones al metamodelo de UML. En la sección 6.2 se presentan las extensiones al metamodelo UML para la aplicación de contratos traslúcidos.
3. **Vista individual:** muestra los requisitos de un solo tema (sin cortes).

5.2 Theme/UML

Pasos del proceso Theme/UML:

1. Diseño de temas.
2. Especificar las relaciones entre los temas.
3. Composición de temas para la verificación de propósitos.

6 Resultados experimentales

En la figura 2 se presenta el diagrama de clases del caso de estudio de un sistema de software con aplicación Domótica, diseñado para demostrar la aplicación y utilidad de los contratos traslúcidos.

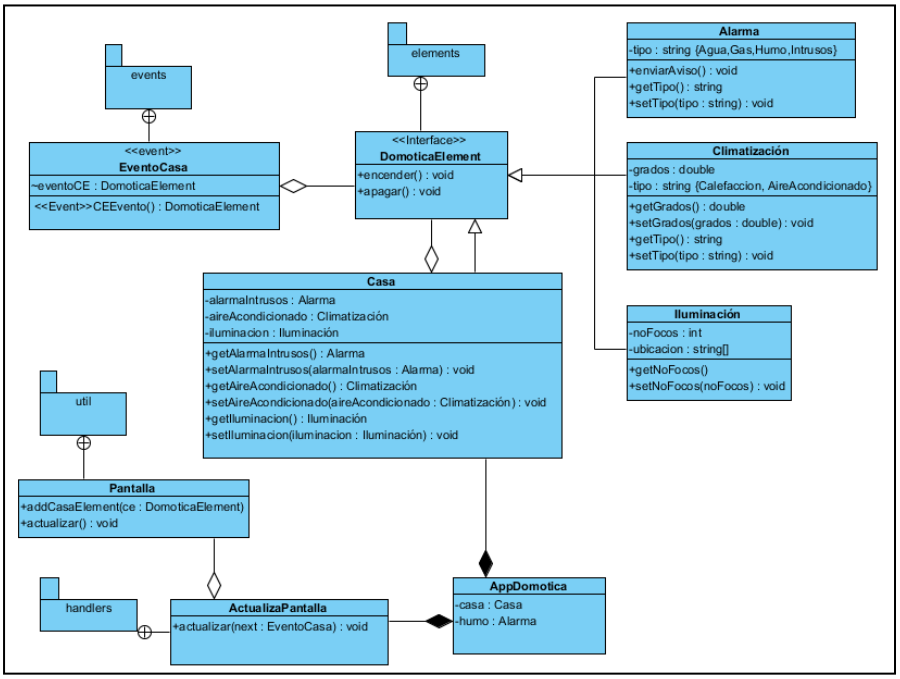

Figura 2. Diagrama de clases para el caso de estudio.

6.1 Requisitos de la aplicación de Domótica

A continuación se listan los requisitos que se implementan en la aplicación Domótica:

R1. El sistema **registra** en pantalla cuando algún elemento de la casa (climatización, iluminación o alarma) es encendido o apagado.
R2. El usuario **enciende** el elemento Climatización, seleccionar el tipo deseado (Calefacción o Aire Acondicionado).
R3. El usuario **enciende** el elemento Iluminación: debe indicar el número de focos y seleccionar la ubicación.
R4. El usuario **enciende** el elemento Alarma: es necesario indicar el tipo de alarma (agua, gas, humo o intrusos).

R5. El usuario **enciende** el elemento Casa que integra todos los elementos deseados a activar al salir de la casa.
R6. El usuario **apaga** el elemento Climatización, elige el tipo a desactivar.
R7. El usuario **apaga** el elemento Iluminación: debe indicar el número de focos y seleccionar la ubicación a desactivar.
R8. El usuario **apaga** el elemento Alarma: debe indicar el tipo de alarma a desactivar.
R9. El usuario **apaga** el elemento Casa para desactivar todos los elementos de la configuración.

En la figura 3 se presenta el diagrama de vistas de acción para la aplicación Domótica. Una vista de acción consiste en 2 elementos: acciones (rombos) y requisitos (rectángulo redondeado). Para desarrollar las diversas vistas, se identifican en el requisito la acción clave y se conecta la acción con el requisito.

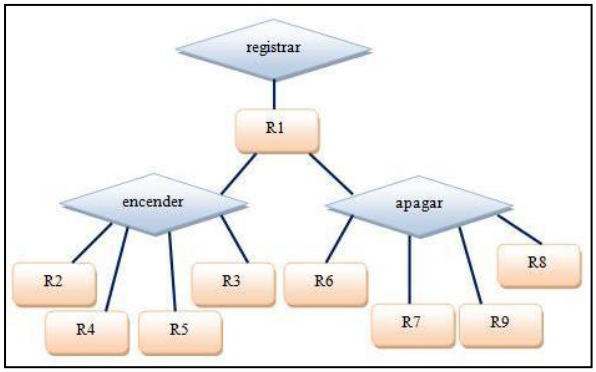

Figura 3. Diagrama de vistas de acción.

En la figura 4 se presenta el diagrama de vistas de corte para la aplicación Domótica. Se observa que entre R1, R5 y R9 existen requisitos comunes (encender y apagar), por lo que **casa** es un tema de corte, es decir el requisito se encuentra disperso en el software y se confunde (interfiere) con el código de otros requisitos.

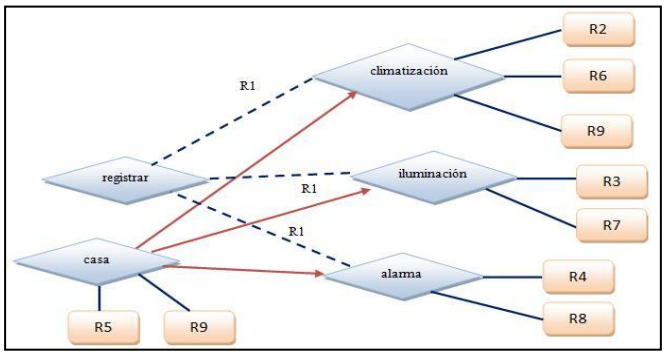

Figura 4. Diagrama de vistas de corte.

6.2 Elementos propuestos en el enfoque de temas para la aplicación de contratos traslúcidos

1. Elegir el tema dominante, es decir aquel tema base que posea una relación del tipo **"has-a"** (agregación o composición), en el caso de estudio el tema **registrar** el cual representará la definición del contrato traslúcido para el evento denominado **EventoCasa**.
2. Al identificar los temas base y de corte: casa, climatización, iluminación y alarma, se propone extender al metamodelo UML con el estereotipo aviso (<<announcement>>) para indicar que en cada clase es necesario refinar el contrato traslúcido y mantener el control de efectos.
3. Se recomienda añadir el estereotipo <<handler>> para indicar la implementación del asunto de corte. En el caso de estudio, la clase **ActualizaPantalla** representa el *handler*, el cuál se encarga de registrar el orden en que recibe los avisos.

6.3 AspectJ vs Ptolemy

En la tabla 1 se presenta la implementación de aspectos de la aplicación Domótica en los lenguajes AspectJ y Ptolemy.

Tabla 1. Cuantificación AspectJ vs Ptolemy

Cuantificación en AspectJ	Cuantificación en Ptolemy
pointcut encendido(): execution(* domotica.*.encender()); (1)	**when** EventoCasa **do** update;
pointcut apagado(): execution(* domotica.*.encender()); (2)	

Como se observa la identificación de puntos de unión en AspectJ se consigue mediante patrones de firmas (1 y 2) y en Ptolemy se realiza mediante la llamada al evento a identificar con las palabras reservadas **when y do**.

Al proceso de conocer el número de eventos a identificar se le conoce como **cuantificación.** Sin embargo AspectJ presenta el problema de la **fragilidad de cortes** debido a que cuantifica mediante expresiones (pseudo) regulares, ya que si se modifica un aspecto o el sistema cambia, el corte no garantiza que cumpla la regla original para identificar los eventos de interés y algunos pueden ser excluidos.

Lo anterior muestra que Ptolemy permite cuantificar de manera inconsciente, al no depender del patrón de firmas, garantizando que el sistema reaccione a los eventos registrados.

7 Trabajos relacionados

En [10] se analiza la complejidad de los temas relacionados con el DbC y la POA. La naturaleza dinámica de los aspectos junto con la inconsciencia limita a las

metodologías existentes del DbC para hacer frente a los programas orientados a aspectos. Un mecanismo del DbC para la POA tiene que abordar la ejecución de contratos (sobre clases y aspectos) así como las consecuencias que deben ser validados entre los contratos sobre aspectos y los contratos en clases. Todas estas ideas se integran en una herramienta llamada CONA, basada en el DbC con aspectos, la cual valida contratos y aspectos en tiempo de ejecución. CONA es una extensión del lenguaje Java y AspectJ. Sin embargo, en este trabajo se reconoce que es posible mejorar el razonamiento modular sobre los aspectos.

En [11] se describe un enfoque orientado a aspectos llamado Theme/UML que extiende a UML para ayudar a los diseñadores en la modularización de asuntos incluyendo cortes. Un tema es cualquier característica, asunto o requisto que tiene que cumplir el sistema. Las extensiones de Theme/UML a UML se centran alrededor de cómo los temas de corte se relacionan entre si y cómo se definen las capacidades de composición. Para proporcionar esta capacidad Theme/UML define un nuevo tipo de relación llamada relación de composición que permite al diseñador identificar aquellas partes de los temas que se relacionan entre si y por lo tanto deberían ser compuestas.

En [12] se aborda que Theme/Doc ofrece vistas de las especificaciones de los requisitos, cuyo propósito es exponer las relaciones entre comportamientos de los requisitos. Este trabajo presenta un caso de estudio que muestra que Theme/Doc es un método eficaz para identificar aspectos en las etapas iniciales del ciclo de vida del software (requisitos).

8 Conclusiones y trabajo a futuro

Para garantizar que el software a desarrollar sea fiable se requiere de mejores prácticas que gestionen de manera adecuada requisitos y asuntos. En la actualidad, existen metodologías como el Enfoque de Temas que apoyan la separación de asuntos (vistas de acción) e identificación de asuntos de corte (vistas de corte) para minimizar el acoplamiento entre ellos. Sin embargo es necesario conservar esa separación conceptual a lo largo del ciclo de vida del desarrollo de software orientado a aspectos, por lo cual es de gran importancia implementar correctamente los contratos traslúcidos desde las etapas de análisis y diseño.

La importancia de un contrato traslúcido radica en que combina las ventajas de la Programación Orientada a Aspectos y el Diseño por Contratos de forma dinámica porque el contrato es flexible y se aplica de acuerdo al evento identificado resolviendo mediante este mecanismo los problemas que presentan algunos Lenguajes Orientados a Aspectos como CaesarJ y AspectJ como la fragilidad de cortes, cuantificación, razonamiento modular y control de efectos.

Los resultados obtenidos permiten detectar las oportunidades que el enfoque de temas ofrece para contribuir a un adecuado tratamiento de requisitos. Como trabajo a futuro se considera establecer una metodología completa y robusta que integre los

conceptos de la POA y el DbC como una alternativa flexible para el desarrollo de software fiable.

Agradecimientos

Este trabajo cuenta con apoyo por parte del Consejo Nacional de Ciencia y Tecnología (CONACyT).

Referencias

1. H. Rajan, G. T. Leavens, R. Dyer, and M. Bagherzadeh, “Modularizing crosscutting concerns with ptolemy,” in Proceedings of the tenth international conference on Aspect-oriented software development companion, ser. AOSD '11. New York, NY, USA: ACM, 2011.
2. I. S. U. of Science and Tecnology, “Ptolemy language,” 2012, http://ptolemy.cs.iastate.edu/docs/.
4. M. Bagherzadeh, H. Rajan, G. T. Leavens, and S. Mooney, “Translucid contracts: expressive specification and modular verification for aspect-oriented interfaces,” in Proceedings of the tenth international conference on Aspect-oriented software development, ser. AOSD '11. New York, NY, USA: ACM, 2011.
4. M. Bagherzadeh, H. Rajan, G. T. Leavens, and S. Mooney, “Translucid contracts for modular reasoning about aspect-oriented programs,” in Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, ser. SPLASH ’10. New York, NY, USA: ACM, 2010
5. B. Meyer, Object Oriented Software Construction. Prentice Hall, 2002.
6. M. Bagherzadeh, G. T. Leavens, and R. Dyer, “Applying translucid contracts for modular re soning about aspect and object oriented events,” in Proceedings of the10th international workshop on Foundations of aspect-oriented languages, ser. FOAL '11. New York, NY, USA: ACM, 2011
7. M. Bagherzadeh, “Enabling expressive aspect oriented modular reasoning by translucid contracts”, in Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, ser. SPLASH '10. New York, NY, USA: ACM, 2010
8. S. Clarke and E. Baniassad, Aspect Oriented Analysis and Design. Addison Wesley Professional, 2005.
9. E. Baniassad and S. Clarke, “Theme: An approach for aspect-oriented analysis and design,” in Proceedings of the 26th International Conference on Software Engineering, ser. ICSE ’04. IEEE Computer Society, 2004.
10. D. H. Lorenz and T. Skotiniotis, “Extending design by contract for aspect-oriented programming,” 2005.
11. E. Baniassad and S. Clarke, “Finding aspects in requirements with theme/doc,” 2004.
12. S. Clarke, “Aspect-oriented design with theme/uml,” Junio 2004.