AspectMDA: Hacia un desarrollo incremental consistente integrando MDA y Orientación a Aspectos

Pablo Amaya Barbosa
Dept. de Informática
Universidad de Extremadura
10071 Cáceres
pabloama@unex.es

Carlos González Contreras
Dept. de Informática
Universidad de Extremadura
10071 Cáceres
carlosgc@unex.es

Juan M. Murillo Rodriguez
Dept. de Informática
Universidad de Extremadura
10071 Cáceres
juanmamu@unex.es

Resumen

El Desarrollo Dirigido por Modelos, y especialmente la propuesta MDA de la OMG, han surgido para hacer frente a ciertos problemas en el desarrollo de software actual. Por ejemplo, a la rapidez con la que un producto software requiere estar disponible para su uso y al alto grado de evolución que poseen los sistemas software actuales. Pero los beneficios prometidos por estas aproximaciones han quedado minimizados en el desarrollo de grandes sistemas complejos debido al tamaño y complejidad que poseen los modelos que describen dichos sistemas. En este punto es donde aparece el Desarrollo de Software Orientado a Aspectos (DSOA) mejorando la comprensión, reutilización y adaptación de los artefactos software modularizando los crosscutting concerns en entidades aisladas y bien identificadas denominadas *aspectos*. Por estos motivos proponemos integrar el DSOA y MDA con el objetivo de suplir las deficiencias de este último. Así, aspectos tales como: seguridad, replicación, restricciones de tiempo real, etc., serán modelados por grupos de trabajo especializados en cada uno de esos aspectos de forma independiente y descentralizada a través de todo el framework MDA. Además, adaptamos una herramienta con el fin de chequear la consistencia entre los diferentes modelos que constituyen el sistema a un mismo nivel de abstracción, establecer las relaciones de composición entre dichos modelos, mejorar la trazabilidad entre distintos niveles de abstracción y controlar el impacto de cambios en el desarrollo de estos sistemas con MDA.

1. Introducción

El Desarrollo Dirigido por Modelos (Model-Driven Development –MDD-) [8] es una disciplina que actualmente está acaparando gran atención e interés en la comunidad investigadora. El objetivo primordial es restar preponderancia a la etapa de codificación del ciclo de vida para depositar más responsabilidad en el modelado. Ello permite a los diseñadores centrarse en la semántica del sistema a construir, abstrayéndose de todos los detalles de la implementación relativos a la plataforma subyacente.

Prueba de ello es el auge de la aproximación Model-Driven Architecure (MDA) [11] de la OMG, que establece tres niveles de abstracción sobre los que se diseña e implementa un sistema software: Computational Independent Model (CIM), Platform Independent Model (PIM) y Platform Specific Model (PSM). Entre cada nivel de abstracción se hallan las transformaciones, otro de los mecanismos claves de MDA y MDD [8], cuyo propósito es establecer reglas de transformación entre elementos de un modelo de abstracción fuente hacia otro más refinado o más abstracto.

Ahora bien, cuando MDA es aplicado en el desarrollo de grandes sistemas complejos, los beneficios prometidos por éste framework disminuyen sustancialmente (trazabilidad, evolución, mantenimiento, etc.). Este problema surge porque los modelos que especifican el sistema se vuelven excesivamente grandes, complejos y monolíticos [18], provocando que dichos modelos sean difíciles de mantener, evolucionar, extender, reutilizar, etc. Además, las transformaciones entre los distintos modelos de abstracción se vuelven muy complejas, excesivamente grandes y poco reutilizables. En este ámbito, la trazabilidad de requisitos a través de los modelos situados a diferentes niveles de abstracción resulta prácticamente imposible.

Por otra parte, el DSOA ha surgido para llevar a todas las fases del ciclo de vida software las aportaciones proporcionadas por la Programación Orientada a Aspectos [6][7] . Este paradigma

supone un avance en la modularización del software. Más concretamente permite aislar en artefactos (denominados aspectos) aquellas propiedades del sistema cuya especificación queda dispersa por el sistema y cuyo aislamiento resulta imposible con las técnicas convencionales de modelado.

De este modo, la orientación a aspectos está aportando soluciones a los problemas de MDA mencionados anteriormente [1][10], ya que aplicando técnicas de Separación de Aspectos a cada nivel de abstracción se pueden obtener modelos mejor modularizados, y por lo tanto, más manejables, mantenibles, reutilizables, etc. En el punto de trabajos relacionados se discuten las diferentes aproximaciones y las diferencias mayores con la aquí presentada.

Nuestra propuesta se basa en la idea de especificar diferentes modelos que corresponden con distintos aspectos del sistema. De este modo, los niveles MDA son especificados por especialistas en diferentes facetas del sistema que modelan los *aspectos* de la misma y los mantienen separados durante todo el framework [14].

En este trabajo, son dos los objetivos primordiales a cubrir. El primero es modelar esos aspectos por cada equipo de trabajo desde el CIM hasta el PSM con el mínimo de comunicación posible entre ellos. Así, se podrán definir procesos de desarrollo incrementales, iterativos y paralelos en el contexto de MDA. El segundo objetivo es adaptar la herramienta denominada xlinkit [3] con el fin de chequear y validar los modelos que constituyen un determinado nivel de abstracción, soportar la trazabilidad de elementos entre distintos niveles y controlar el impacto de un cambio de requisitos dentro del framework.

El resto del artículo está organizado de la siguiente forma: en el apartado 2 se presenta la propuesta de forma global con un ejemplo y como adaptamos xlinkit para nuestros objetivos; en el apartado 3 se expone las mejoras introducidas en la trazabilidad y facilidad de evolución en MDA; en el apartado 4 se discute como lograr un proceso de desarrollo incremental y consistente con nuestra propuesta; en el apartado 5 se presentan los trabajos relacionados; y en el punto 6 las conclusiones y posibles líneas de trabajo futuro.

2. Desarrollo Concurrente de Aspectos en el contexto de MDA

Este apartado esta organizado en dos sub-apartados: el primero presenta una visión general de la propuesta de modelado con un ejemplo; y el segundo muestra cómo se aplica xlinkit para chequear la consistencia entre los modelos desarrollados por los distintos equipos de trabajo debido a posibles incongruencias que puedan surgir entre dichos modelos.

2.1. Modelado de Aspectos en MDA

Como ya se ha descrito, el primer objetivo de este trabajo se basa en la idea de dar soporte a aquellos *crosscutting concerns* que se han detectado en el sistema con el fin de aislarlos, modelarlos de forma separada desde el CIM hasta el PSM y componerlos a nivel de código. Note que los aspectos serán identificados en la etapa de requisitos [5] y que la descomposición de los mismos determinará los aspectos de un sistema a modelar con nuestra propuesta (esta cuestión está fuera del alcance de los objetivos del trabajo). Además, nuestro propósito y especial objetivo en este trabajo es que dichos modelos puedan ser desarrollados por diferentes equipos de trabajo de forma descentralizada, concurrente y consistente con un mínimo de comunicación entre ellos [4].

Asimismo, es necesario que entre los diferentes modelos de un mismo nivel de abstracción se establezcan sus relaciones de composición, con el fin de especificar la semántica compartida entre todas las facetas del sistema, detectar conflictos entre los modelos, validarlos o integrarlos en un todo. Dichas relaciones se especifican de forma externa a los modelos, con el fin de que los grupos de trabajo se centren en sus aspectos a desarrollar siendo inconscientes de los demás aspectos del sistema.

Para cumplir con lo mencionado anteriormente tenemos que tener una serie de factores en cuenta:

- Los modelos se podrán desarrollar en ámbitos físicamente distribuidos por la red.
- Especificar las relaciones de composición con un mecanismo externo debido a las limitaciones de UML para establecer las relaciones entre elementos distribuidos por la red.
- Posibilidad de componer el sistema en su conjunto a nivel de modelo con fines de análisis, detección de conflictos, visón global, etc.

- Especificar puntos concretos de chequeo dentro del marco de trabajo para mantener la consistencia entre los modelos distribuidos.

Por lo tanto, una característica determinante en la propuesta es cómo modelar las relaciones de composición entre los modelos distribuidos para que cumplan con los anteriores requisitos. Para ello, se ha determinado establecer las relaciones de composición en formato XML independientemente de los modelos. Esto será explicado de forma más detallada en el apartado 2.2 Xlinkit.

La Figura 1 muestra el nivel PIM de la propuesta modelando un pequeño caso de estudio de un sistema de información e-government. Dicho estudio trata sobre el protocolo administrativo sancionador en la Junta de Extremadura. El sistema consta de un conjunto de expedientes que son creados cada vez que un ciudadano comete una falta administrativa. Así, dicho expediente puede incurrir en una multa hacia el ciudadano según lo estime el instructor del caso. En el ejemplo hemos identificado tres casos de uso: Crear un nuevo expediente, recuperar un expediente tras haber aplicado una multa y autenticar al usuario que realiza algunas operaciones.

Como Ivar Jacobson explica en [9], en un proceso de desarrollo OO, un caso de uso es realizado por un conjunto de clases que colaboran para implementarlo, pero también, esas clases ayudan a implementar la funcionalidad de otros casos de uso. Esto causa dispersión de esa funcionalidad por todas las clases y enmarañamiento dentro de las clases cuando implementan partes de varios casos de uso. Esto provoca que añadir un nuevo caso de uso al sistema implique muchos cambios en diseño e implementación. Dicho problema también ha sido identificado como una falta de alineamiento entre requisitos, diseño y código por Subject-Oriented Desing (SOD) [18]. Esta propuesta propone mantener cada uno de los requisitos del sistema diseñados e implementados en entidades independientes (*subjects*). Por dicho motivo utilizamos Subject Oriented Modeling (SOM)[16] con el fin de diseñar e implementar cada caso de uso de forma independiente y aislada.

En la Figura 1 aparecen dos aspectos del sistema que son modelados con el estereotipo <<viewPIM>> (2) del elemento *model* de UML2 [12]. Asimismo, dichos aspectos son modelados en el nivel CIM y PSM con el estereotipo <<viewCIM>> y <<viewPIM>> respectivamente. Los aspectos anteriormente citados son: autentificación de usuarios (en paquete *Security* (4)) y otro que concierne a gestión de expedientes (en paquete *Expedient* (3)). El primero contiene un patrón de composición (8) [16] que modela el aspecto de autentificación de usuarios en un sistema. En el paquete *Expedient* se hallan dos *subjects* que diseñan dos casos de uso: NewExpedient (7) y RecovExpedient (1). Como se pueden observar son paquetes con el stereotipo <<*subject*>>. Así, cada uno de ellos modelan las

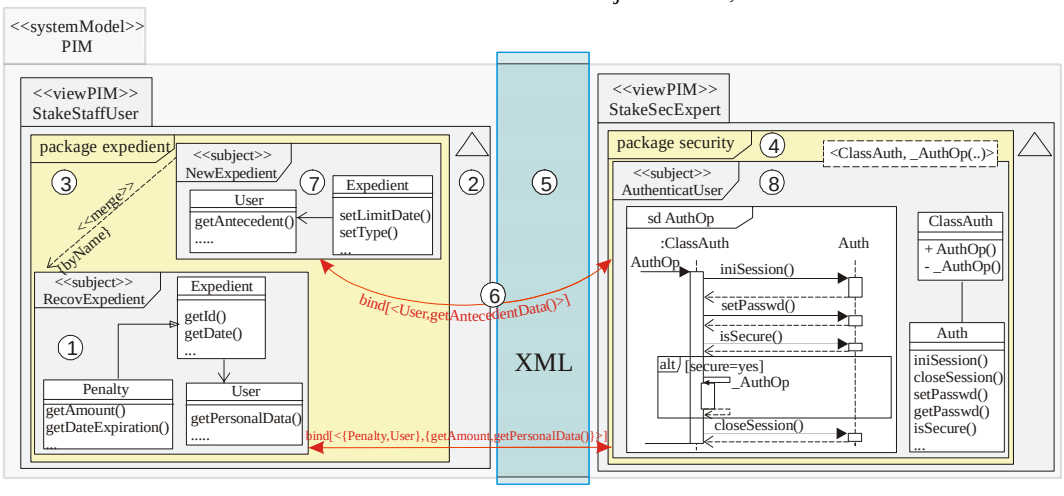

Figura 1.Nivel PIM con dos aspectos utilizando Subject-Oriented Modeling

clases necesarias para proporcionar la funcionalidad del caso de uso que diseñan. Como se puede observar, entre dichos *subjects* aparece una relación de composición (<<merge>> {byName}), en este caso indicando que se hará una composición de mezcla entre los dos sujetos a través de los nombres de las clases [16]. Ahora bien, como se puede observar, existen dos relaciones de tipo *bind* (6) que relacionan elementos entre el *subject* AuthenticaUser (8) con NewExpedient (7) y RecovExpedient (1). Esta relación es utilizada por los patrones de composición y tiene el siguiente significado: si observamos la relación (6) *bind[<User,getAntecedentData()>]*, especifica que el método getAntecedentData() de la clase User va modificar su comportamiento con el especificado por el diagrama de interacción AuthOp del *subject* de autentificación.

Todos estos *subjects* se transformarán en uno o varios *subjects* en el nivel PSM, y a partir de estas se puede generar el código.

Un detalle importante de la propuesta es la utilización de XML para relacionar los viewPIM *(5)* y que en este ejemplo son las dos relaciones *bind* en rojo. Se ha optado por establecer las relaciones de composición entre los elementos de los distintos modelos de este modo por los siguientes motivos principalmente:

- Debido a que UML (la propuesta se basa en este lenguaje) no permite establecer correspondencias entre modelos desarrollados por separado.
- Para habilitar un desarrollo independiente entre los grupos de trabajos que modelan aspectos, para que tengan el mínimo de comunicación y puedan trabajar concurrentemente. De esta forma un modelador-coordinador establecerá las relaciones de composición entre los distintos modelos
- Porque XML es aceptado como el lenguaje de marcado universal para el intercambio de información.
- Y además, xlinkit requiere que todos los documentos a procesar (modelos, reglas y en nuestro caso relaciones de composición) estén en formato XML.

Como hemos indicado anteriormente, hemos adaptado una herramienta denominada xlinkit, cuyo fin es validar los modelos desarrollados por los distintos grupos de trabajo y las relaciones de composición entre los mismos. Xlinkit está basada en XML y administra la consistencia entre documentos heterogéneos distribuidos por la red que describen modelos de Ingeniería del Software [3]. De este modo, los viewModels (viewCIM, viewPIM o viewPSM) descritos en la aproximación son exportados al formato XMI [13] propuesto por la OMG. Así, una vez que los modelos, las relaciones de composición y las reglas de restricciones están en formato XML, se ejecuta la herramienta Xlinkit para que establezca y valide los modelos y relaciones de composición. El proceso y mecanismo de xlinkit es explicado con más detalle en el siguiente apartado.

2.2. Xlinkit

Xlinkit es una propuesta para administrar la consistencia de documentos XML heterogéneos distribuidos por la red que son centrales para el desarrollo de sistemas software [3]. Esta aproximación es utilizada para chequear la consistencia de documentos durante todo el ciclo de vida, tanto a un mismo nivel de abstracción (horizontal) como entre distintos niveles (vertical). Dichos documentos son chequeados contra un conjunto de restricciones implementadas como reglas. Una regla muy sencilla es chequear si una implementación en Java es consistente con su diseño en UML, partiendo de que establezca que para toda clase en el diseño debe existir una clase en la implementación con el mismo nombre.

Esta aproximación está basada en XML, XPath y XLink para la generación de hipervínculos entre documentos distribuidos. La herramienta acepta un conjunto de documentos XML que representan *modelos* junto con otros que contienen las reglas que los elementos de esos modelos deben satisfacer. Como salida produce un informe en XML con parejas o ternas de enlaces a los elementos consistentes e inconsistentes (*LinkBase*). Es decir, si dos elementos de dos modelos satisfacen una determinada regla, LinkBase muestra una terna con hipervínculos hacia la regla y hacia los elementos consistentes.

El propósito original de xlinkit es administrar la consistencia entre modelos, pero en nuestra propuesta es usada y adaptada para algunos propósitos más:

- Chequear la consistencia entre los viewPIMs (y viewPSMs) en un mismo nivel de abstracción junto con las relaciones de

```
<xlinkit:ConsistencyLink ruleid="aspect.xml#/id('r1')">
  <xlinkit:State>consistent</xlinkit:State>
  <xlinkit:Locator
   xlink:href="http://localhost/relationsihps.xml#//relationship[@id='1']"/>
  <xlinkit:Locator
   xlink:href="http://host1/View1.xml#//Model_Management.Package[@xmi.id='4']"/>
  <xlinkit:Locator
   xlink:href="http://host2/View2.xml#//Model_Management.Package[@xmi.id='6']"/>
</xlinkit:ConsistencyLink>
```

c) LinkBase de una relación consistente

```
<globalset id="$view" xpath="//Model_Management.Package[@xmi.id]"/>
<globalset id="$rel"  xpath="//relationship/subject" />
<consistencyrule id="r1">
 <description>
  Todos los elementos especificados en una relación de composición deben existir
 </description>
 <forall var="r" in="$rel">
  <exists var="s" in="$view">
    <implies>
     <equal op1="$r/@name" op2="$s/Foundation.Core.ModelElement.name/text()" />
     <equal op="id($s/Foundation.Core.ModelElement.Stereotype/
                   Foundation.Extension_Mechanisms.Stereotype/
                   @xmi.idref)[Foundation.Core.ModelElement.name
                   /text()]"
            stereo="subject" />
    </implies></exists></forall></consistencyrule>
```

b) Restricción simple en XML

```
<relationship type="bind" id="1">
  <subject name="newExpedient" type="subject">
    <concreteElement class="User">
     <op name="getAntecedentData" />
    </concreteElement>
  </subject>
  <subject name="AuthenticaUser" type="pattern"/>
</relationship>
```

a) Relación de composición en XML

Figura 2. Ficheros XML

composición (chequeo horizontal). Para ello desarrollamos una serie de reglas que validan o identifican conflictos en las relaciones de composición entre esos modelos. Por ejemplo, en una relación de composición merge entre dos subjects, estos no pueden tener dos métodos idénticos en dos clases a mezclar ya que producirían un conflicto semántico de composición.

- Chequear las consistencias entre un modelo y su transformación en otro más refinado o más abstracto (chequeo vertical).
- Utilizar LinkBase como documento para *enlazar* las relaciones de composición entre los viewModels, como fuente para dar soporte a la trazabilidad de los elementos que constituyen los diferentes modelos abstractos de un aspecto, y con el fin de controlar el impacto de un cambio.

Siguiendo el ejemplo de la Figura 1, el primer paso para que xlinkit procese los viewPIMs es especificar las relaciones de composición en XML. La especificación es una trascripción propia que hemos realizado partiendo de las relaciones `<<bind>>`, `<<merge>> y <<override>>` [16]. En la Figura 2.a se muestra la relación `bind[<User,getAntecedentData()>]` en XML (Figura 1 (6)).

El segundo paso es crear el conjunto de reglas que chequeen y establezcan las distintas relaciones especificadas en el XML de composición. En la figura 2.b se observa una regla muy sencilla que valida la relación *bind* anterior, comprobando que los elementos especificados en la relación existan en los dos modelos y cuyo estereotipo sea *subject*. Otra regla es, por ejemplo, validar que los parámetros y elementos que enlaza el bind son compatibles entre sí y que no se haya omitido ninguno.

El tercer paso es tener accesibles los viewPIMs en la red en formato XMI. Así, con los tres documentos descritos anteriormente ya se

puede ejecutar xlinkit con el propósito de procesar los modelos y las relaciones de composición contra el conjunto de reglas especificadas. Tras esto, la herramienta genera el fichero LinkBase en formato XML dividido en dos áreas:

- La primera contiene aquellos elementos que son consistentes entre los viewPIMs (una terna *subject-relación-subject*).
- La segunda contiene los elementos inconsistentes que han violado alguna de las reglas contra las que fueron chequeados.

A dicho documento se le da un valor añadido, ya que se usa como fuente para la composición, debido a que almacena referencias (Xlink) a elementos de distintos modelos y a las relaciones de composición que los relacionan. En la Figura 2.c se observa un LinkBase sencillo mostrando que dos elementos ([@xmi.id='4'] y [@xmi.id='6'] -- Figura 1 (7) y (8)) de los dos viewPIMs y la relación de composición ([@id='1'] – Figura 1 (6)) son consistentes con la regla de consistencia 'r1'.

Además, para generar el código de cada modelo de aspectos se utiliza Hyper/J [7]. Para ello, el LinkBase y el XML de composición son utilizados para derivar las relaciones de composición entre los distintos Hyperslices e Hypermodules con el objeto de obtener la implementación del sistema en su conjunto. En [17] se detalla como transformar cada *subject* y relación de composición a Hyper/J.

En los dos próximos apartados se explica como utilizar el LinkBase para automatizar la trazabilidad y controlar la propagación de un cambio de requisitos en el marco del trabajo aquí presentado.

3. Trazabilidad y control de propagación de cambio

Como se ha indicado anteriormente, xlinkit proporciona dos tipos de chequeos: verticales y horizontales. Así, con el fin de obtener el máximo grado de consistencia posible, es interesante establecer en que momentos del desarrollo con MDA se van ha realizar los chequeos. Además, en grandes sistema complejos no es posible, ni a veces deseable, realizar un chequeo de consistencia de forma global a todo el sistema [2].

Los momentos que han sido identificados en el framework MDA son muy concretos:

- Cuando un viewModel está listo para transformarlo en otro, es conveniente chequearlo contra el resto de modelos de su mismo nivel de abstracción.
- Tras la transformación, los modelos origen y destino son chequeados para comprobar que ésta se ha realizado correctamente.
- Cuando en un nivel determinado se introduzca un nuevo viewModel (ya sea por transformación, inserción desde repositorio, etc.), hay que chequearlo contra los modelos con los que se relacione en dicho nivel.

En la Figura 3 se muestran los LinkBase generados a partir de los puntos de ejecución de xlinkit determinados en la propuesta.

Estos documentos sirven de entrada a un algoritmo que traza cualquier elemento de modelado dentro del framework. Por ejemplo, suponemos que en nuestro desarrollo hemos optado por implementar el sistema en el lenguaje Java. En un determinado momento es detectado un fallo de diseño en una clase Java, entonces lo ideal es trazar de "donde proviene esa clase", es decir, que elementos del PIM y PSM (incluso CIM) son los "causantes" de esa clase. Procesando los documentos Linkbase de una forma sencilla se realiza esa trazabilidad de arriba hacia abajo y vicecersa. El proceso se basa en tomar del LinkBase el elemento fuente que queremos trazar y obtener los elementos que forman su pareja de consistencia *vertical*, y así sucesivamente hasta el nivel que se desee llegar.

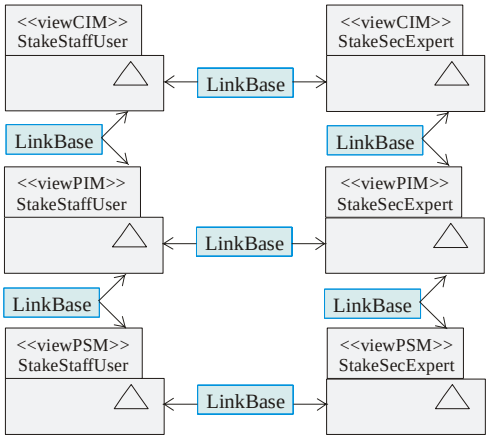

Figura 3. LinkBase generados

Note que tanto el Subject-Oriented Modeling y xlinkit contribuyen a la mejora de la trazabilidad: el primero porque mejora el alineamiento entre los niveles MDA, y entonces el diseño de los *concerns* no es esparcido por todo el modelo, sino que es encapsulado en entidades asiladas y bien identificadas; y el segundo porque a través de la generación de los enlaces entre elementos del PIM y el PSM proporcionado por los Linkbase, se da soporte a un algoritmo tanto para trazar *concerns* (*subjects*) como cualquier otro elemento de modelado.

Además, la tarea de controlar la propagación de un cambio está muy relacionado con lo enunciado en el párrafo anterior. Si una vez que el sistema está en desarrollo necesitamos hacer un cambio en requisitos, en el diseño, o en un elemento simplemente, sería muy deseable saber de antemano qué elementos de los niveles inferiores y superiores se verán afectados por ese cambio. Gracias a que se puede trazar en las dos direcciones, simplemente con procesar los LinkBase obtenemos qué elementos de otros niveles podrían verse afectados y por lo tanto ejecutar el cambio, posponerlo o cancelarlo.

El mismo procedimiento puede emplearse para trazar y controlar los cambios de forma horizontal. En este caso se procesan los LinkBase que relacionan distintos viewModels dentro de un mismo nivel de abstracción, es decir los LinkBase horizontales.

Esta característica es muy importante para el mantenimiento y evolución del sistema, ya que la tarea de modificar clases, relaciones, decisiones de diseño, etc., pueden ser automatizadas y controladas en gran medida por la utilización conjunta de MDA, DSOA y xlinkit. Es decir, se pueden desarrollar vistas completas (viewCIM, viewPIM y viewPSM) que modelan *aspectos* concretos (facilitando la identificación y modificación de entidades relacionadas con una determinada faceta del sistema); con xlinkit chequear y validar los modelos del sistema, mejorar la trazabilidad y controlar la propagación de un posible cambio en el sistema; y con las transformaciones automatizar y agilizar cambios en el sistema. Por lo tanto, podemos afirmar que nuestra propuesta integra estas tecnologías o aproximaciones de una forma muy adecuada y beneficiosa para el Desarrollo Dirigido por Modelos.

4. Hacia un desarrollo incremental consistente.

Otro de los beneficios de integrar Subject-Oriented Modeling con MDA es una mejora en el proceso de desarrollo incremental de grandes sistemas complejos. Esto es debido a que esta aproximación de modelado orientado a aspectos permite añadir o modificar el comportamiento y estructura a entidades previamente modeladas de forma *aditiva en vez de invasiva*. Por ejemplo, una vez que se han modelado los tres niveles de abstracción del aspecto de seguridad (viewCIM, viewPIM y viewPSM) en nuestro caso de estudio, puede que el sistema requiera un cambio en la especificación de requisitos: “El control de acceso se realizará sobre un flujo seguro con SSL”. Esta modificación conllevará crear un nuevo caso de uso que extiende (<<extend>>) al anterior de seguridad. Asimismo, este cambio implica modificar entidades del PIM y PSM, pero en este caso realizándose de forma aditiva. Es decir, sobre “AuthenticateUser” (Figura 1 (8)) se aplicará un subject con el nuevo comportamiento de seguridad sin modificar el ya existente. Y para el PSM exactamente lo mismo. Esta forma de realizar cambios de forma aditiva en un proceso de desarrollo software supone una mejora en el proceso de desarrollo incremental, en la evolución y mantenimiento del sistema software.

Además, el comportamiento de xlinkit ante estos tipos de cambios aditivos encaja de manera perfecta. Xlinkit permite realizar un análisis incremental de la consistencia, es decir, extrae las diferencias entre un modelo XMI antes y después de su modificación, analizando sólo aquellos elementos que pudieran haber quedado inconsistentes tras las modificaciones. Por lo tanto, como el cambio introducido en el sistema está bien identificado y aislado, sólo chequeará el nuevo *aspecto* y los elementos relacionados con él.

5. Trabajos Relacionados

La propuesta más similar a la nuestra es [1] y proponen usar las propuestas de modelado orientado a aspectos en el nivel PSM. El motivo que los autores argumentan para ello es que dichas propuestas son muy dependientes de la plataforma (algo de lo que no estamos de acuerdo). Así, para

elevar el nivel de abstracción de modelado sugieren utilizar distintos Domain-Specific Languages (DSL) para cada aspecto que se modela. El problema es que para cada aspecto nuevo en el sistema hay que utilizar un nuevo DSL (basado en la extensión del meta-modelo UML o perfiles), y por lo tanto los desarrolladores deben trabajar con varios lenguajes dentro de un mismo nivel de abstracción. Además, es una aproximación que está muy orientada a una arquitectura Web.

En [10] integran aspectos y MDD para facilitar la trazabilidad, reusabilidad y evolución en un sistema software. Esta propuesta presenta un meta-modelo basado en plantillas para la separación de aspectos. La cuestión más criticable es que las propias plantillas acoplan aspectos unos a otros.

En [15] presentan una aproximación de modelado genérica similar a la aquí utilizada. Ésta también podría ser utilizada como mecanismo de modelado dentro de nuestra propuesta, pero hace una separación clara entre modelo núcleo con la funcionalidad al que se le aplican los demás modelos basados en aspectos. Esta visión está muy influenciada por AspectJ [6], mientras nuestro trabajo está más cercano a la separación de asuntos multi-dimensional [7], puesto que nosotros abogamos que es una aproximación a la separación de aspectos más flexible y potente.

Ivar Jacobson analiza en [9] los problemas de enmarañamiento y dispersión en los diagramas de componentes en el desarrollo de software dirigido por casos de uso. Él propone solucionar estos problemas usando separación de asuntos multi-dimensional estableciendo como dimensiones casos de uso y clases, aunque no menciona nada de transformaciones, composición, relaciones estructurales, etc.

Nuestra propuesta tiene ciertas características similares a Theme [4] debido a que es una evolución del SOD, aunque en ese trabajo los requisitos son especificados usando "Action Views" y diseñados posteriormente con subjects denominados *themes*. La mayor diferencia es que no proponen nada sobre transformaciones de los modelos ni de los *themes*, las etapas que presentan son análisis y diseño sin hacer hincapié en posibles etapas intermedias o refinamientos de los modelos. Además, la identificación de conflictos en la composición se realiza sobre todo el sistema global y no sobre elementos bien localizados como se ha comentado anteriormente.

6. Conclusiones y Trabajo Futuro:

En este trabajo hemos presentado un marco de trabajo en MDA aplicando Desarrollo de Software Orientado a Aspectos, proponiendo a los aspectos de un sistema como diferentes modelos manteniéndolos separados desde el CIM hasta el PSM. Además, adaptando e integrando la herramienta xlinkit se desarrollan los viewModels por diferentes equipos de trabajo de forma concurrente, independiente y consistente. Asimismo, la propuesta integra un mecanismo *flexible y externo* (xlinkit) para dar soporte a la trazabilidad dentro de MDA de arriba-abajo y viceversa, la cuál también es mejorada debido a la mejora de alineamiento entre modelos inducido por la utilización de SOM. Incluso el mantenimiento y evolución del sistema pueda hacerse de una forma controlada a través de la identificación de los elementos que pueden verse afectados tras un cambio en el sistema. Además, debido a la utilización del modelado orientado a aspectos conseguimos que dichos cambios sean añadidos de forma aditiva e incremental.

Actualmente estamos trabajando sobre un repositorio de viewModels que cubran todo el framework MDA. Es decir, tener en un repositorio los modelos de un aspecto que cubren los tres niveles de abstracción para reutilizarlos en distintos sistemas dentro del mismo dominio.

Como ya se ha indicado anteriormente, también estamos realizando un catálogo con reglas sobre restricciones de composición de subjects con el fin de ejecutar un fuerte chequeo a nivel de modelo. Esto validará la composición a ese nivel de abstracción para que el código generado a partir de dicho modelo no tenga problemas de consistencia.

Agradecimientos

Este trabajo ha sido financiado por el proyecto de la Junta de Extremadura 2PR04B011 y del MCYT TIC2002-04309-C02-01

Referencias

[1] A.M. Reina, J. Torres, and M. Toro. Towards developing generic solutions with aspects. Workshop in Aspect Oriented Modelling held in conjunction with the UML 2004 Conference, oct 2004.

[2] B. Nuseibeh, S. Easterbrook, and A. Russo. Leveraging Inconsistency in Software Development. IEEE Computer, 33(4):24-29, April 2000.

[3] C. Nentwich, W. Emmerich, A. Finkelstein and E. Ellmer, "Flexible Consistency Checking," ACM Transactions on Software Engineering and Methodology, vol. 12, pp. 28-63, 2003.

[4] D. L. Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. In Communications of ACM, Vol. 15, issue 12. ACM, December 1972.

[5] E. Baniassad and S. Clarke. Theme: An approach for Aspect-Oriented Analysis and Design. In Proceedings of the 26th ICSE, 2004.

[6] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.M. Loingtier, and J. Irwin. Aspect-oriented programming. In ECOOP'97—Object- Oriented Programming, 11th European Conference. LNCS 1241, pages 220- 242, 1997.

[7] P. Tarr, H. Ossher, W. H. Harrison, and S. S. Jr. N degrees of separation: Multi-dimensional separation of concerns. In Proceedings of the ICSE, pages 107–119. IEEE Computer Society Press,1999.

[8] IEEE Software. Special issue on Model-Driven Development. Volume 20, number 5. September/October 2003.

[9] Ivar Jacobson: Use Cases and Aspects – Working Seamlessly Together. In Journal of Object Technology, vol. 2, no. 4, July-August 2003, pp. 7-28. http://www.jot.fm/issues/issue_2003_07/column1

[10] Kulkarni, Vinay and Reddy, Sreedhar: Integrating aspects with Model Driven Software Development. In International Conference on SERP'03. Las Vegas, USA. June 2003

[11] OMG. MDA Guide V1.0.1. Document -- omg/03-06-01

[12] OMG. UML 2.0 Superstructure. Document -- ptc/04-10-02

[13] OMG. XMI 2.0. Document- formal/03-05-02

[14] P. Amaya, C. González, J.M. Murillo. “MDA and Separation of Aspects: An approach based on multiple views and Subject Oriented Design” in Workshop on Aspect Oriented Modelling held in conjunction with the AOSD 2005 Conference, mar 2005.

[15] R. B. France, I. Ray, G. Georg, and S. Ghosh. An aspect-oriented approach to design modeling. IEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, 151(4), August 2004.

[16] S. Clarke. "Extending standard UML with model composition semantics" in Science of Computer Programming, Volume 44, Issue 1, pp. 71-100. Elsevier Science, July 2002.

[17] Siobhán Clarke, Robert J Walker. "Separating Crosscutting Concerns across the Lifecycle: From Composition Patterns to AspectJ and Hyper/J". [TCD-CS-2001-15], Trinity College, Dublin and UBC-CS-TR-2001-05, University of British Columbia. May 2001

[18] S. Clarke, W. Harrison, H. Ossher, P. Tarr. "Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code". In Proceedings of OOPSLA) Denver, Colorado U.S., November 1999.