Información Tecnológica
Vol. 23(1), 3-12 (2012)
doi: 10.4067/S0718-07642012000100002

Aplicación de la Modelación Orientada a Aspectos

Cristian L. Vidal[(1)], Dannia D. Hernández[(2)], Cristian A. Pereira[(3)], María C. Del Río[(1)]
(1) Escuela de Ingeniería Informática Empresarial, Fac. de Ciencias Empresariales, Univ. de Talca Campus Lircay, Avenida Lircay S/N, Talca-Chile (e-mail: cvidal@utalca.cl; cdelrio@utalca.cl)
(2) Dirección de Acreditación, Universidad de Magallanes, Avenida Bulnes 01855, Punta Arenas-Chile (e-mail: dannia.hernandez@umag.cl)
(3) Epsilonforce, Gabriela Mistral 0850, Depto. 1108, Temuco-Chile (e-mail:cpereira@epsilonforce.com)

Recibido Jun. 08, 2011; Aceptado Ago. 01, 2011; Versión Final recibida Ago. 16, 2011

Resumen

Se aplica la Modelación Orientada a Aspectos a la primera etapa del ciclo de desarrollo de software, que es la especificación y análisis de requerimientos, con el uso de Diagramas de Casos de Uso UML para la identificación temprana de aspectos de aplicación como herramienta de modelación. Dada la importancia de la detección temprana de Aspectos y la actual relevancia del Desarrollo de Software Orientado a Aspectos (DSOA), se describe y resalta la importancia de utilizar una herramienta estándar de modelación en la detección temprana de aspectos. Se analiza también los potenciales beneficios de la aplicación de este concepto de desarrollo de software y se concluye que la identificación temprana de incumbencias permite un mayor conocimiento del sistema a desarrollar, lo que debe traducirse a un desarrollo de software más eficiente.

Palabras clave: modelación orientada a aspectos, programación, incumbencias transversales, ingeniería de software.

Application of Aspect Oriented Modeling

Abstract

The Aspect Oriented Modeling is applied to the first phase of software development cycle, which is the specification and the analysis of requirements, with the use of UML Use Cases Diagrams for the early identification of application aspects as modeling tool. Due to the importance of early aspects detection, and the current relevance of the Aspect Oriented Software Development concept (AOSD), the importance of using a modeling standard tool to detect them is described and highlighted. The potential benefits of applying this concept to software development are also analyzed. It is concluded that the early identification of tangling allows a better identification of the system to be developed making software development more efficient.

Keywords: aspect oriented modeling, programming, aspects, crosscutting concern, software engineering

INTRODUCCIÓN

Modelación es un elemento fundamental para alcanzar el éxito en el proceso de desarrollo de Software. De esta forma, en el actual paradigma de Desarrollo de Software Orientado a Aspectos (DSOA), Modelación Orientada a Aspectos juega un rol de absoluta importancia para conseguir software de calidad (Jacobson et al., 2004; Nakajima et al., 2004; Rosenhainer et al., 2004; Skipper, 2004; Tabares et al., 2008; Tesendic et al., 2007; Xu et al., 2005; Yu et al., 2005).

Como (Pressman, 2001) indica, el Proceso de Desarrollo de Software es divisible en 4 etapas importantes: Especificación y Análisis de Requerimientos, Diseño, Implementación y Mantención. En la historia se han utilizado diversos patrones o paradigmas de desarrollo de software. Estos paradigmas siempre han buscado la simplificación del problema global de construcción de software a través de la modularización o división de tareas (Pressman, 2001). Este principio es una constante desde el paradigma de desarrollo de software estructurado hasta el paradigma de desarrollo de software orientado a objetos, donde siempre se ha buscado la simplificación del problema global en problemas relativamente más simples. En la historia del proceso de desarrollo de software, a pesar de que las metodologías de división de problemas simplifican el problema global, existen elementos en los módulos de la solución no posibles de modularizar con el uso de técnicas clásicas de desarrollo de software (Jacobson et al., 2004). Se indica que el desarrollo de software con metodologías de desarrollo de software tradicionales divide el problema verticalmente, y cuando aparece un elemento horizontal común, no es posible lograr una completa aislación de este elemento. Justamente, el paradigma DSOA permite la división vertical y horizontal de los elementos presentes en el desarrollo de software, los cuales deben ser identificadas y modeladas en las primeras etapas de este proceso para un desarrollo de software de calidad. Los elementos identificados en el desarrollo de software se denominan globalmente Incumbencias, donde los elementos verticales son las Incumbencias o Aspectos base y los elementos horizontales son Incumbencias o Aspectos cruzados (Tabares et al., 2008). Cuando el Desarrollo de Software Orientado a Objetos (DSOO) ha alcanzado una completa difusión en el entorno de desarrollo de software, y UML es el Lenguaje de Modelación Orientada a Objetos por defecto, es necesario indicar que DSOA es una extensión de DSOO, y justamente se ha trabajado en la adaptación de herramientas de DSOO para DSOA, entre ellas UML (Jacobson et al., 2004). Este trabajo presenta la aplicación de Casos de Uso UML para la Modelación Orientad a Aspectos (MOA) de un sistema de software para una biblioteca, con la identificación y modelación de los elementos verticales y horizontales (Aspectos), y así resaltar la simplicidad en las siguientes etapas de DSOA, dada la familiarización con el problema por la identificación temprana de Aspectos. En la sección de Conclusiones, se indican los principales logros a obtener con el uso de DSOA y se dan a conocer ideas de trabajo de Modelación Formal Orientada a Aspectos (MFOA).

DESARROLLO DE SOFTWARE ORIENTADO A ASPECTOS

Incumbencias representan diferentes temas o asuntos del problema base en el proceso de desarrollo de software. Toda aplicación tendrá Incumbencias base para funciones específicas, pero también surgen Incumbencias cruzadas presentes en Incumbencias base. Ejemplos de Incumbencias cruzadas son seguridad, distribución, persistencia, replicación, sincronización, entre otros. Logrando una separación de Incumbencias, se disminuye la complejidad a la hora de trabajar con ellas, y es posible cumplir con requerimientos relacionados con la calidad como adaptabilidad, mantenibilidad, extensibilidad y reusabilidad (Laddad, 2003).

La solución que propone la comunidad de DSOA es modularizar las Incumbencias horizontales o cruzadas, las que se denominan Aspectos cruzados. Globalmente, se busca la separación de Incumbencias y establecer una relación entre ellas de manera transparente para las Incumbencias base como ilustra la Figura 1 (Jacobson et al., 2004; Tabares et al., 2008). Existen términos relevantes a ser considerados y utilizados en el paradigma DSOA (Laddad, 2003; Tabares et al., 2008) que se describen a continuación. Gracias a estos elementos es posible definir y trabajar con los Aspectos cruzados presentes en el desarrollo de aplicaciones software. Como ejemplos clásicos de Incumbencias cruzadas se pueden mencionar: transacciones, seguridad, y logging.

1. Puntos de Unión (JoinPoint): Se refiere, globalmente, a los elementos donde puede ser incluido algún elemento o comportamiento adicional.

2. Punto de Corte (PointCut): Elementos dentro de los Aspectos cruzados que determinan si el cruzamiento es o no efectivo. Si se agrega o no algún elemento o comportamiento adicional.

3. Aviso o Sugerencia (Advise): Corresponde al elemento o comportamiento adicional a ser incluido en el Punto de Corte.

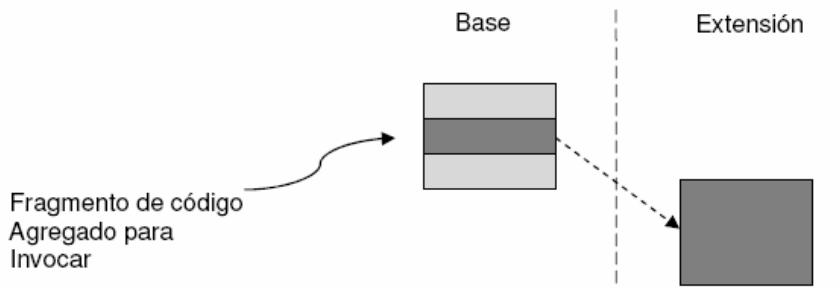

Fig. 1: Separación y relación de Incumbencias Base y de Extensión.

Es necesario enfatizar que DSOA se establece para vencer las limitaciones del paradigma de DSOO, donde no es posible lograr una completa y absoluta modularización en el desarrollo de aplicaciones. La modularización convencional de DSOO, clases y objetos, tienen la inhabilidad de separar o individualizar los Aspectos cruzados (Jacobson et al., 2004).

Es importante señalar que requerimientos e Incumbencias no son lo mismo. Desarrollar un sistema de software involucra una especificación de requerimientos los cuales son entonces afinados en etapas de diseño e implementación. De esta forma, requerimientos son solo parte del ciclo de desarrollo de software. Una Incumbencia representa algo de importancia a más alto nivel: usuarios y clientes. Requerimientos son simplemente especificación de Incumbencias (Jacobson et al., 2004). En general, por cada Incumbencia, existirán varios requerimientos asociados para su clarificación. Esto permite generalizar el término de Incumbencia a Aspectos. En otras palabras, cada módulo se denomina Incumbencia o Aspecto. La separación de Incumbencias es una labor presente en cada una de las etapas o fases del proceso de desarrollo de software.

El nacimiento del paradigma DSOA se debe principalmente a la aparición y masificación del paradigma de Programación Orientada a Aspectos (POA) donde uno de sus principales lenguajes de lo constituye Aspect/J como una extensión del lenguaje Java. Sin embargo, actualmente ya existen plug-in que trabajan con el paradigma de Orientación a Aspectos para la mayoría de los lenguajes de programación convencionales, desde C, PHP, Python, a .NET y Java (Laddad, 2003; Tabares et al., 2004).

DIAGRAMAS DE CASOS DE USO

Diagramas de Casos de Uso son fundamentales para la identificación de requerimientos en el DSOO. Como DSOA es una extensión de DSOO, es posible la aplicación de Diagramas de Casos de Uso en DSOA (Jacobson et al., 2004). Además, un ejemplo de aplicación de Diagramas de Casos de Uso en la modelación de procesos de negocio se presenta en (Sanchis et al., 2009). Es necesario indicar, actualmente se trabaja en lograr la generación automática de aplicaciones de software o la generación automática de código a partir de especificaciones de requerimientos mediante diagramas de Casos de Uso UML (Zapata et al., 2010). Como se ha señalado previamente, Aspectos cruzados se refiere a Incumbencia cruzadas, y un Caso de Uso permite la representación de Incumbencias cruzadas cuando su implementación involucra un número de clases mayor a uno. Justamente, si un Diagrama de Caso de Uso esté sujeto al uso de variables

que se cruzan entre diferentes Casos de Uso, estas variables son modeladas como flujos alternativos o relaciones de extensión.

Se debe considerar que Diagramas de Casos de Uso no representan solamente una técnica asociada a requerimientos. Diagramas de Casos de Uso son una técnica de Ingeniería de Software para dirigir completamente el ciclo de vida del desarrollo de software, donde es relevante la identificación de Incumbencias para su modelación, implementación y validación. Justamente, Diagramas de Casos de Uso permiten expresar Incumbencias de usuario, y proveer una temprana validación de las mismas. De esta forma, Diagramas de Casos de Uso UML son un útil medio para dirigir el DSOA, donde el desarrollo incremental mediante Casos de Usos representa un tradicional enfoque de desarrollo de software.

En la Figura 2 se presenta un ejemplo simple de modelación usando Diagramas de Casos de Uso de un sistema informático con uso de logging. Como se aprecia en esta figura, la relación de extensión en Diagramas de Casos de Uso, es útil para la modelación de Incumbencias cruzadas.

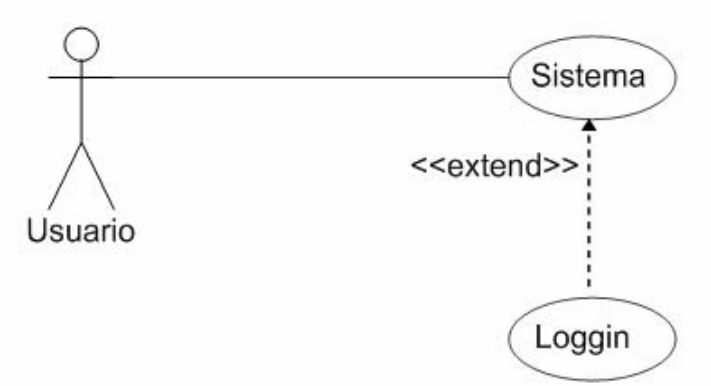

Fig. 2: Diagrama de Casos de Uso de Sistema con uso de Loggin.

Después de revisar la Figura 2, donde existe una modelación de Incumbencia cruzada, es claro que Diagramas de Casos de Uso UML permiten la capturar temprana efectiva de Incumbencias desde los requerimientos de usuario. Como señala (Jacobson et al., 2004), con el uso Diagramas de Casos de Uso es posible conocer y avanzar en la comprensión del comportamiento deseado del sistema ante las potenciales acciones del usuario y las posibles respuestas del sistema. Una idea adicional que será utilizada en la modelación de caso de estudio es que la representación de relaciones de inclusión (incluye) usando relaciones de extensión para soportar completamente el paradigma de DSOA. También es posible la utilización de relaciones de herencia (Jacbson et al., 2004; Pressman, 2001), las cuales también pueden ser modeladas como relaciones de extensión.

Definitivamente, un Diagrama de Casos de Uso es una excelente herramienta para dirigir el proceso de desarrollo de software dado que permite representar el comportamiento de un sistema desde la perspectiva del usuario. Gracias a esto, es posible un temprano test y validación de requerimientos de usuario, es posible la identificación de clases de dominio del sistema y las relaciones entre ellas (trabajo conjunto), y habilita la identificación de casos de uso críticos y de alta prioridad. Además, Diagramas de Casos de Uso son útiles para el desarrollo de software en general, no sólo con una visión de Orientación a Objetos.

Tal y como señalan (Jacobson et al., 2004; Li, 2009; Rosenhainer, 2004), para una modelación en DSOA con Diagramas de Casos de Uso es necesario la presentación de un diagrama de Clases de Dominio del sistema para tener una mayor comprensión del ámbito del sistema, y se requiere la identificación de la variabilidad de acciones para su representación en Diagrama de Casos de Uso con la utilización de relaciones de inclusión. En la siguiente sección se presenta un Caso de Estudio para un sistema de software con la identificación de requerimientos, la identificación de los módulos del sistema y el correspondiente diagrama de clases de dominio, y con la aplicación de Diagramas de Casos de Uso en DSOA.

CASO DE ESTUDIO

Como caso de estudio para su modelación usando Diagramas de Casos de Uso en DSOA, se utiliza el sistema de biblioteca BISIS descrito en (Tesendic, 2007), el cual es altamente aplicable en ámbitos similares donde sea necesario implementar un sistemas de biblioteca. De acuerdo a la referencia citada, BISIS tiene los siguientes módulos:

1.- Información de Usuarios de Biblioteca: Es necesario un conjunto de datos de usuario al momento de registrar un usuario. Este conjunto de datos es definido por cada biblioteca. Algunos datos deberían ser obligatorios al momento del registro de un nuevo usuario y esta característica de obligatoriedad es también definida por la biblioteca, el cual difiere entre bibliotecas, de manera que es necesario establecer la obligatoriedad de datos separadamente.

2.- Datos de Membresía: Es necesario registrar datos asociadas a la renovación de membresía de usuario así como también proveer notas respecto a la expiración de membresía. Hay diferentes tipos de membresía y categorías de usuario, ambos definidos por la biblioteca. Un tipo de membresía determina la duración de la misma, mientras que una categoría de usuario regula el número de préstamos activos o simultáneos que un usuario puede realizar, así como también el período máximo o lapso de préstamo. Un impuesto o costo de membresía de usuario depende de la categoría de usuario y del tipo de membresía definidos por la biblioteca. Por cada registro o renovación de membresía, es también necesario registrar el miembro de la biblioteca que realiza la operación (librero), así como la biblioteca asociada.

3.- Información de uso de Material de Biblioteca: Estos datos son relativos a los préstamos y retornos de material de biblioteca a usuarios. En contacto con el usuario, el librero necesita tener la siguiente información: información de material a prestar, recordatorio de posible atraso en la entrega, así como también el historial de préstamos de usuario. Por cada préstamo, retorno o renovación de préstamo, es necesario registrar el librero y la biblioteca asociada.

4.- Recordatorios Enviados a Usuarios de Biblioteca: Es necesario el envío periódico de recordatorios vía e-mail a los usuarios con atrasos en devolución. Hay diversos tipos de recordatorios y el contenido de ellos difiere de librería a librería. Así es necesario habilitar cambios en el contenido de recordatorios. En contacto con el usuario, el librero necesita obtener la información del recordatorio de atraso recientemente enviado al usuario. Impresión anual de registro de recordatorios de atrasos que la biblioteca preserva en su archivo histórico debería ser posible obtener.

5.- Reportes Estadísticos: Una posibilidad de generar cierto conjunto de reportes estadísticos acerca de usuarios y uso de la biblioteca es esperado. Los reportes incluyen lo siguiente: usuarios registrados; usuarios registrados según tipo de membresía, categoría y sexo; historial de préstamos por cada usuario; historial de préstamos por cada elemento de biblioteca (libro o revista); libros más leídos; lectores más activos. Es necesaria una posibilidad de búsqueda de usuario de acuerdo a todos los criterios anteriores. Esto debería ser posible en cada una de las bibliotecas.

6.- Funcionamiento de Interfaz de Usuario: En orden para alcanzar un trabajo más eficiente con atención especial de usuarios es necesario disminuir el número de operaciones que un librero debería realizar cuando procesa una petición de usuario, así como también una rápida respuesta del sistema de software.

7.- Registro Histórico de Datos de Usuario: También, de manera de realizar un trabajo más eficiente con el usuario, es necesario remover los datos de usuarios que no renuevan su membresía, esto es, archivarlos a ellos.

Dado estos requerimientos, en la Figura 3 presenta el Diagrama de Clases de Dominio de BISIS. El objetivo principia de un Diagrama de Clases de Dominio de BISIS es tener una comprensión mayor para analizar el sistema y diseñar el Diagrama de Casos de Uso asociado. Un Diagrama de Clases de Dominio es una abstracción, ya que no representa un elaborado Diagrama de Clases ya que no es necesario representar los atributos y métodos de Clases, así como la multiplicidad y tipo de relaciones entre clases.

Dada el primer análisis de BISIS con la modelación de Diagrama de Clases de Dominio, es posible identificar sus módulos o casos de uso principales, los que se señalan a continuación:

1.- Módulo de bibliotecas donde se realizan operaciones sobre el conjunto de bibliotecas.

2.- Módulo de usuarios donde se realizan operaciones sobre el conjunto de usuarios de cada biblioteca.

3.- Módulo de categorías, donde es posible establecer las categorías de usuarios de cada biblioteca.

4.- Módulo de movimientos donde se realizan operaciones de préstamo, renovación, devolución, o cancelación de deuda en cada biblioteca.

5.- Módulo de ítems donde es posible realizar operaciones sobre el conjunto de ítems de cada biblioteca.

6.- Módulo de Recordatorios donde el sistema genera y envía mediante correo electrónico recordatorios de devolución a los usuarios con atraso de entrega en cada biblioteca.

7.- Módulo de Reportes donde se accede a reportes estadísticos de BISIS.

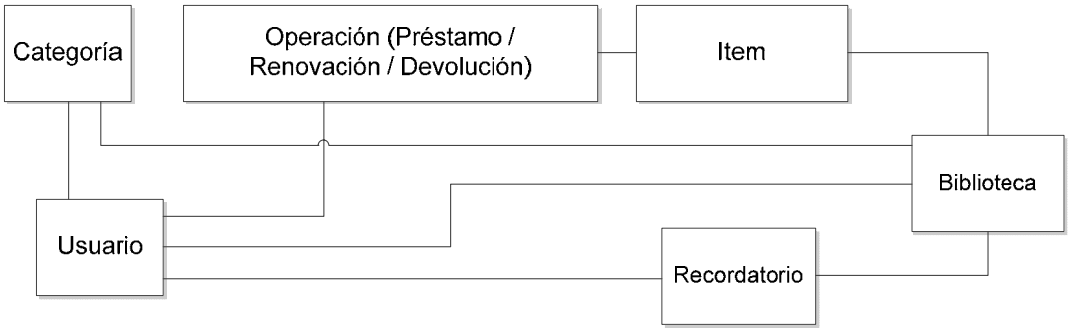

Fig. 3: Diagrama de Clases de Dominio de BISIS.

A continuación, en la Figura 4, se presenta y analiza el Diagrama de Casos de Uso de BISIS. Es necesario indicar que se consideran tres actores principales, Usuario, Librero y Administrador, los cuales tienen acceso a ciertos módulos de BISIS. Administrador tiene acceso al módulo de Ítems, al módulo de Bibliotecas y al Módulo de Reportes, mientras que Librero y Usuario tienen acceso a Información de Usuario, Membresía y Movimientos. Nótese que se aprecian y modelan Incumbencias cruzadas de Loggin en Casos de Uso “Validación 1”, “Validación 2” y “Validación 3”, para validar Usuarios, Libreros y Administradores, respectivamente. Además se modelan como Incumbencias cruzada el Caso de Uso “Ingreso de Datos Obligatorios” para Caso de Uso “Información de Usuario”, y el Caso de Uso de “Categorías” que es una extensión de “Membresía”. En Caso de Uso “Movimientos” se aprecia una relación de herencia con los Casos de Uso “Préstamo”, “Renovación” y “Devolución”, la que podría ser modelada como una relación de extensión para así realizar de manera completa una Modelación Orientado a Aspectos. En la Figura 5 se presenta una descripción del Caso de Uso “Información de Usuario”, y en la Figuras 6, 7 y 8 una descripción de sus Aspectos Cruzados.

CONCLUSIONES

Se ha comprobado empíricamente la importancia y necesidad de una correcta modelación en el proceso de desarrollo de software para las diversas metodologías de desarrollo, donde DSOA no es una excepción. Es importante señalar que la utilización de este paradigma de desarrollo de software permite a los desarrolladores ganar tiempo gracias a la identificación de incumbencias. Se espera que la identificación temprana de incumbencias permita un mayor conocimiento del sistema a desarrollar, lo que debe traducirse a un desarrollo de software más eficiente.

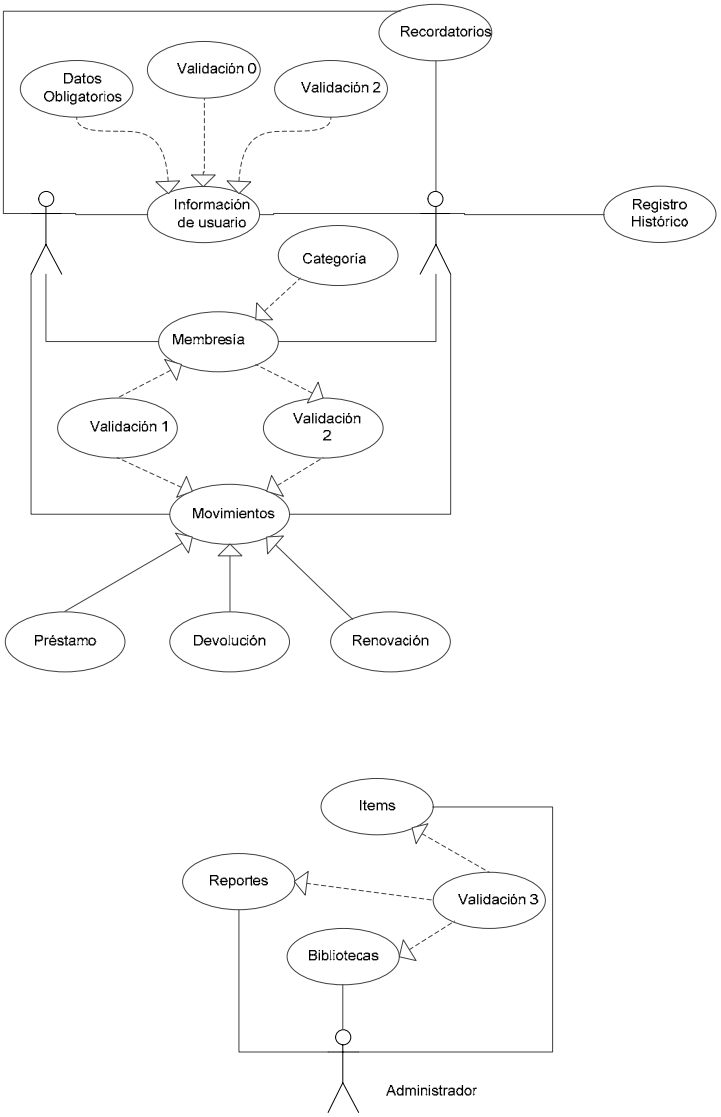

Fig. 4: Diagrama de Casos de Uso de BISIS.

Caso de Uso: Información de Usuario

Flujo Básico: El Caso de Uso comienza cuando un usuario desea registrarse en la biblioteca, y hay un librero para atenderlo.

1. El Librero ingresa al sistema.
2. El Usuario facilita su información.
3. El Usuario es Registrado.
4. El Caso de Uso termina.

Flujos Alternativos:
…
Puntos de Extensión:
E1. Ingreso de Librero. Este punto de extensión ocurre en la paso 1.
E2. Datos de Usuario. Este punto de extensión ocurre en la paso 2.
E3. Validación de Usuario. Este punto de extensión ocurre en la paso 2.

Fig. 5: Descripción Textual Caso de Uso Información de Usuario BISIS.

Caso de Uso: Validación 2

Flujo Básico: Esta extensión ocurre cuando el Librero desea entrar al Sistema, para verificar su autenticidad.

1. El Librero presenta su Id de Usuario.
2. El Librero presenta su Contraseña.
3. Si es un Librero válido el Caso de Uso termina.
4. Si no es Librero válido, no se puede continuar.

Puntos de Corte de Extensión:

punto de corte de extensión → Ingreso de Librero.

Fig. 6: Descripción Textual Caso de Uso Validación 2 BISIS.

Caso de Uso: Validación 0

Flujo Básico: Esta extensión ocurre en el registro de un nuevo usuario, para verificar su no existencia en el sistema.

1. El Librero ingresa información clave del usuario.
2. Si existe ya un Usuario registrado con esa información, no se puede continuar.
3. Si Usuario no existe, el Caso de Uso termina.

Puntos de Corte de Extensión:

punto de corte de extensión → Validación de Usuario.

Fig. 7: Descripción Textual Caso de Uso Validación 0 BISIS

Caso de Uso: Datos Obligatorios

Flujo Básico: Esta extensión ocurre cuando el Usuario debe ingresar información que es obligatoria en BISIS.

1. Librero solicita información obligatoria al Usuario.
2. Si esta información es entregada completamente, Librero la Ingresa y Caso de Uso termina.
3. Si no se entrega completamente información obligatoria, no se puede continuar.

Puntos de Corte de Extensión:

punto de corte de extensión → Validación de Usuario.

Fig.8: Descripción Textual Caso de Uso Datos Obligatorios BISIS.

Con la utilización de la extensión de diagramas de Casos de Uso UML, Lenguaje de Modelación propio de DSOO, es posible la identificación temprana de todas las características de la aplicación a desarrollar, elementos horizontales y verticales. La dedicación de mayor tiempo en etapas de modelación de software, con la identificación y modelación temprana de aspectos de la aplicación, debe involucrar una exitosa implementación. Es destacable además la afirmación de que las relaciones de Inclusión y Agregación en Diagrama de Casos de Uso UML pueden ser modeladas mediante relaciones de Extensión. De esta forma es posible utilizar el paradigma de Desarrollo de Software Orientado a Aspectos en etapas tempranas de Desarrollo de Software mediante Diagramas de Casos de Uso UML. Sólo habría que incluir los cambios realizados a la notación textual de la descripción de un Caso de Uso UML en DSOA.

Es necesario indicar que si bien Diagramas de Casos de Uso UML permiten una Modelación Orientada a Aspectos sin mayores cambios al forma tradicional de aplicación y uso de este lenguaje de modelación, el paradigma DSOA no es completamente respetado, ya que en cada Caso de Uso base es necesario indicar explícitamente los puntos de extensión lo cual es una de las cosas que el DSOA propone evitar. Esto no ocurre en los lenguajes de programación orientados a Aspectos como Aspect/J o el lenguaje de especificación formal orientado a Aspectos AspectZ, donde cada elemento o unidad base no se preocupa o no tiene conocimiento de los elementos que lo extienden.

Como trabajo futuro, se contempla la comparación empírica de una aplicación orientada a aspectos utilizando modelación clásica, versus el resultado de implementar la misma aplicación con la aplicación de Modela Formal Orientada a Aspectos. Se pretende utilizar la extensión realizadas a lenguajes formales para la Modelación Formal Orientada a Aspectos, Alloy (Nakajima et al., 2004) y Aspect-Z (Yu et al., 2005), respectivamente.

REFERENCIAS

Jacobson, I. y Ng, P., Aspect Oriented Software Development with Use Cases, 1ª edición, Addison Wesley Professional, Nueva York, USA (2004).

Laddad, R., AspecJ in Action, Practical Aspect-Oriented Programming, Manning Publications Co., Londres, Inglaterra (2003).

Li, G., Identifying Crosscutting Concerns in Requirement Specifications - A Case Study, Tesis de Magister en Ciencias, Queen's University, Kingston, Ontario, Canada (2009)

Nakajima, J. y Tamai, T., Lightweight Formal Analysis of Aspect-Oriented Models, Actas de The 5th Aspect-Oriented Modeling Workshop In Conjunction with UML 2004, 120-127, Lisboa, Portugal, 11 al 15 de Octubre (2004).

Pressman, R., Ingeniería del Software. Un enfoque práctico, 5ª Edición, McGraw Hill (2001).

Rosenhainer, L., Identifying Crosscutting Concerns in Requirements Specifications, Actas de OOPSLA Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop, 130-138, Vancouver, Canada, October (2004).

Sanchis, R., Poler, R. y Ortiz, A., Técnicas para el Modelo de Procesos de Negocio en Cadenas de Suministro, Revista Información tecnológica, 20(02), 29-40 (2009).

Skipper, M. C., Formal Models for Aspect-Oriented Software Development, Tesis de Doctorado, Colegio Imperial de Londres, Londres, Inglaterra (2004).

Tabares, M. S., Alferez, G. H. y Alferez, E. M., El desarrollo de software orientado a aspectos: un caso práctico para un sistema de ayuda en línea, Revista avances en sistemas e informática, 05(02), 61-68 Jun, (2008).

Tesendic, D., A Database Model for Library Material Usage, Revista Novi Sad J. Math., 37(01), 155-162 (2007).

Xu, W. y Xu, D., A Model-Based Approach to Test Generation for Aspect-Oriented Programs, Actas de First Workshop on Testing Aspect-Oriented Programs, WTAOP'05, 30-38, Chicago, USA, Marzo (2005).

Yu, H., Liu, D., Yang, J. y He, X., Formal Aspect-Oriented Modeling and Analysis by Aspect-Z, Actas de 17th International Conference on Software Engineering and Knowledge Engineering, SEKE'2005, 124-132, Taipei, Taiwan, Republic of China, 14 al 15 de Julio (2005).

Zapata, C. M., Chavarra, J. J., Una Mirada Conceptual a la Generación Automática de Código, Revista EIA, (13), 143-154 (Julio 2010).