Validación de Patrones de Diseño de Comportamiento a través de Perfiles UML: Observer, un caso de estudio

Alberto Cortez [1,2], Claudia Naveda [1,2], Ana Garis[3], Daniel Riesco[3]

[1] Instituto de Investigaciones, Universidad del Aconcagua, Mendoza (MZ), Argentina

[2] Instituto de Informática, Universidad de Mendoza, Mendoza (MZ), Argentina

[3] Universidad Nacional de San Luis, San Luis (SL), Argentina

Resumen. *Los patrones de diseño describen soluciones a problemas recurrentes en la ingeniería de software, proporcionando un importante instrumento para la reutilización de software. En particular, los Patrones de Diseño de Comportamiento (según la clasificación Gof), abordan la especificación de las características dinámicas. Tanto los perfiles UML (por sus siglas en inglés, Unified Modeling Language) como el lenguaje OCL (por sus siglas en inglés, Object Constraint Language) se pueden utilizar como mecanismos para la formalización de los patrones de diseño de comportamiento. El presente trabajo presenta un enfoque para la validación de patrones de comportamiento haciendo uso de perfiles UML y lenguajes OCL, en diagramas estáticos y dinámicos UML.*

1 Introducción

Los patrones de diseño especifican la experiencia de diseñadores expertos, ayudando a los desarrolladores de software en la documentación del software y transferencia de conocimientos, proporcionándoles un vocabulario común. Uno de los catálogos más conocidos acerca de los patrones de diseño es presentado por el "*Gang of Four*" (GoF) por Gamma (1994). Se presentan 23 patrones de diseño siguiendo dos criterios principales de clasificación: alcance y finalidad. El alcance se refiere a su aplicación en clases u objetos; y la finalidad establece una clasificación en creacional (que crean los objetos), estructural (que componen las estructuras) y comportamiento (que representan la interacción entre los objetos).

La formalización de los patrones permite definir en forma precisa su semántica. Se han propuesto diferentes enfoques con el fin de especificarlos formalmente y, por tanto, de mejorar su uso, aplicación y validación: Taibi (2007), Lagarde et al. (2007), Debnath, et al. (2006), Dye (2006), Flores et al. (2006), Dae–Kyoo et al. (2007).

Un enfoque de formalización es el uso de perfiles UML y existen algunos trabajos como el de Debnath et al. (2006), en los que se propone una Arquitectura de Perfiles UML de Patrones de Diseño (APPD).

APPD como se muestra en la figura 1, es una arquitectura de tres capas: en el Nivel 0 se definen las características comunes para todos los perfiles de patrones, en el Nivel 1 se introducen diferentes perfiles siguiendo la clasificación GoF; es decir, un perfil estableciendo respectivamente, las características estructurales, creacionales, de comportamiento, de clase, y de objeto; y en el Nivel 2, los perfiles individuales

elaborados por cada patrón. Si bien la arquitectura establece que debe existir en el Nivel 1 un perfil UML para patrones de comportamiento, el mismo no fue especificado, como así tampoco los perfiles particulares de patrones de comportamiento en el Nivel 2. La arquitectura no incluye características dinámicas de los patrones de diseño, sino aspectos estructurales, limitando su aplicación sólo a diagramas de clases UML.

En anteriores trabajos, se reformuló APPD para la incorporación de patrones de comportamiento y se mostró cómo implementarlo con herramientas UML existentes en el mercado. Se cambió el concepto teórico de operaciones propuestas en Debnath et al. (2006), no existe una herramienta estándar UML que verifique el resultado de este tipo de operaciones OCL dentro de un perfil por estereotipos que representan los aspectos generales de cada patrón.

El presente trabajo amplía su implementación respecto de trabajos anteriores de Cortez et al. (2012), para permitir mayor portabilidad hacia herramientas estándar UML. Se propone la construcción de nuevos patrones y la simplificación de su representación, a través de la importación de perfiles.

A partir de los perfiles importados, se pueden utilizar estereotipos de distinto nivel, reutilizándose definiciones y restricciones. Se redujo la estructura del perfil de comportamiento, que contiene un estereotipo representativo de la dinámica de los objetos. Con el objetivo de chequear la consistencia entre los diagramas de clase y secuencia, se formularon en este último perfil reglas de consistencia entre dichos diagramas (El lenguaje UML no posee reglas de consistencia entre diagramas). Es decir, la estructura lograda contiene la especificación, no sólo de aspectos estructurales, sino también de comportamiento. La propuesta es explicada a través de un caso de estudio. Se especifican restricciones para la validación de patrones de diseño de comportamiento, utilizando modelos UML estáticos y dinámicos. Se recurre a la herramienta RSA (por sus siglas en inglés, Rational Software Architect), para cumplir este objetivo.

La organización del trabajo es la siguiente: en la Sección 2 se presentan los trabajos relacionados. En la Sección 3 se explica la especificación de Patrones de Diseño de Comportamiento, se mencionan los estereotipos diseñados que caracterizan los aspectos estructurales para el Nivel 0 de APPD, los estereotipos que caracterizan el comportamiento en el Nivel 1 y los estereotipos que expresan las características del patrón *Observer* en el Nivel 2, aplicando restricciones OCL sobre el mismo. En la sección 4 se presenta un caso de estudio aplicado al patrón *Observer*, validando algunas restricciones OCL que chequean la consistencia. En la Sección 5 se exponen las conclusiones y líneas de trabajo futuro.

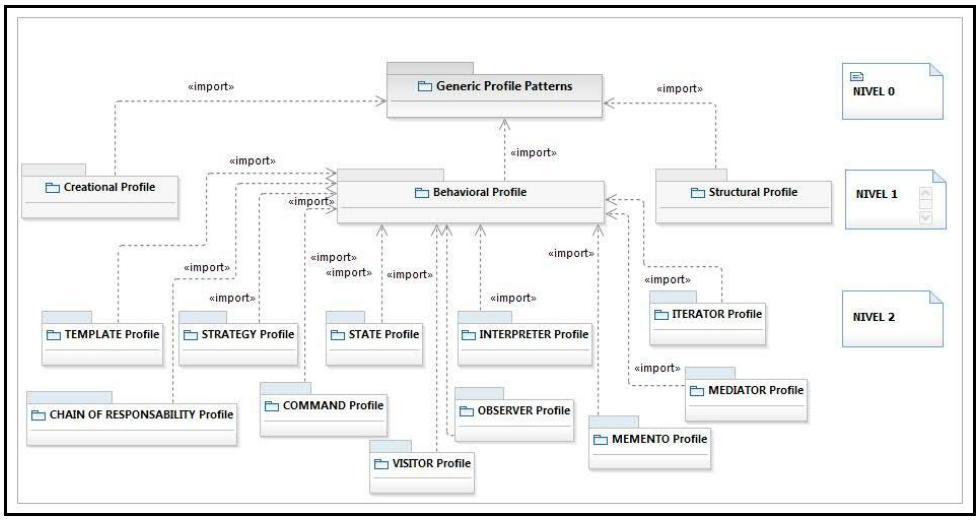

Fig. 1 : Arquitectura de Perfiles para Patrones de Diseño (APPD).

2 Trabajos Relacionados

Dae-Kyoo et al. (2007) presenta el lenguaje RBML para definir las propiedades estructurales y de comportamiento. RBML se basa en roles y especializa el metamodelo de UML. Dye (2006) propone un lenguaje de especificación formal de los patrones de diseño combinando mecanismos de extensión UML para reconocer patrones en diagramas de clases UML. Ambos enfoques (RBML y el lenguaje propuesto por Dye) definen nuevos lenguajes que se extienden del metamodelo UML, pero no tienen en cuenta los perfiles de UML. Definen los estereotipos y las restricciones OCL para la validación; sin embargo, ya que consideran UML versión 1.3 no utilizan perfiles UML (los perfiles UML se introducen a partir de la especificación UML versión 2.0). También, el enfoque de métodos formales se ha aplicado a los patrones de diseño como en Dye (2006) y Flores et al. (2006).

Taibi (2007) define el lenguaje BPSL (por sus siglas en inglés, Balanced Pattern Language Specification) para la formalización de la estructura (a través de lógica de primer orden) y el comportamiento (a través de lógica temporal) de los patrones. Dado que estas características se modelan por diferentes lógicas, no existe una herramienta capaz de validar ambas en un mismo modelo.

El enfoque propuesto por Meyer et al. (2006), establece el criterio de componentización. Dicho criterio establece que si un patrón de diseño puede convertirse en un componente reutilizable, optimiza el proceso de desarrollo. La desventaja de la metodología de Meyer es que restringe las herramientas de diseño solo al lenguaje Eiffel para implementar la componentización y validación de los patrones.

3 Especificaciones de Patrones de Diseño de Comportamiento

APPD permite el uso de perfiles UML para la definición y la visualización de patrones de diseño en los modelos UML, como se muestra en la figura 1. Sin embargo, con el fin de adoptar dicha arquitectura para la validación de los patrones de diseño de comportamiento, se requiere la inclusión de una serie completa y precisa de especificaciones en los diferentes niveles, a saber, los niveles 0, 1 y 2. Los metamodelos

para formular los perfiles fueron extraídos de especificaciones de OMG (por sus siglas en inglés, Object Managment Group) UML Superstructure, (2011). Las mismas tienen dos capítulos dedicados respectivamente a Clases e Interacciones, que en este trabajo representan aspectos estáticos y dinámicos.

Nivel 0

Las características estructurales requeridas en los patrones de diseño se especifican en este nivel. Y para su validación se determinan las propiedades de los patrones en diagramas estáticos de UML. Los estereotipos incorporados en este nivel extienden las metaclases del metamodelo de clases de la superestructura de UML:

patternPackage: Extiende la metaclase Package lo que permite definir restricciones en el contexto paquete. Se validan características propias de un patrón, como por ejemplo las relaciones que deben existir entre los elementos.

patternClassifier: Extiende la metaclase Classifier y representa una clase abstracta o interface.

patternClass: Extiende la metaclase Class y describe una clase de tipo concreto que además implementa las operaciones de un tipo *patternClassifier*.

patternProperty: Extiende la metaclase Property y valida la corrección de sus características de una clase.

patternOperation: Extiende la metaclase Operation y describe operaciones implementadas en una clase.

Por ejemplo, cada participante de un patrón es *patternClassifier* o *patternClass*; en el primer caso, el participante debe ser una clase abstracta o una interfaz, y en el segundo caso, una clase concreta. El estereotipo *patternOperation* se refleja en diagramas estáticos, pero sirve para chequear los mensajes de invocación en diagramas dinámicos. Los mensajes deben derivarse de operaciones estereotipadas.

Nivel 1

En esta capa de la arquitectura se incorpora las especificaciones de los perfiles de comportamiento, que validan la coherencia entre los diagramas estáticos y dinámicos; en particular, para los diagramas de clase y secuencia UML. El estereotipo *patternInteraction,* extiende la metaclase *Interaction* del metamodelo de comportamiento. De esta manera se valida que los elementos de una interacción estén correctamente construidos, por ejemplo los mensajes y líneas de vida. El estereotipo *patternOccurrence* verifica la consistencia entre ocurrencias de especificación, operaciones y mensajes. Para aplicar los perfiles de Nivel 1 es obligatoria la existencia de un diagrama de clase estereotipado por Nivel 0.

Nivel 2

Se incluye en este nivel perfiles de patrones de diseño particulares. Se importan para su construcción los perfiles del Nivel 0 y Nivel 1. Concretamente, los estereotipos del Nivel 2 heredan las propiedades de los estereotipos y sus restricciones asociadas de los niveles 0 y 1.

En particular presentamos la especificación del patrón *Observer,* con el propósito de mostrar el enfoque. Dicho patrón define una dependencia entre objetos: un objeto sujeto mantiene una asociación con objetos dependientes, denominados observadores. Los observadores están registrados (attached), entonces pueden ser notificados cuando un cambio ocurre. Por lo tanto, cada observador actualiza su estado.

La descripción de este patrón de diseño de comportamiento, que consiste en características estáticas y dinámicas, se puede ver en la Figura 2. El estereotipo *ObserverConsistency* sirve como contexto para validar los aspectos generales del patrón, por ejemplo la existencia de relaciones: asociación, agregación y composición. Los estereotipos *attach, setState, update, getState* y *detach* sirven para validar los aspectos dinámicos.

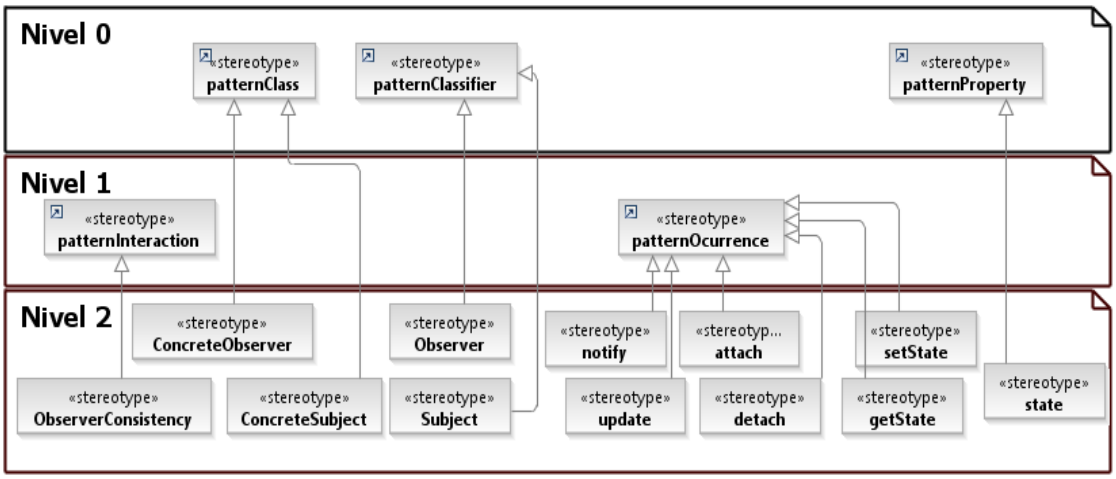

Fig. 2: Características estructurales y dinámicas del patrón Observer.

A continuación, se muestran algunas restricciones, construidas para esta arquitectura en lenguaje OCL y su descripción.

Restricción 1: Asociación entre los estereotipos ConcreteSubject y ConcreteObserver

```
Context PerfilNivel2::ObserverConsisstency
Association.allInstances ().memberEnd->
   select (m|m.isNavigable()=true
      and m.type.getAppliedStereotypes()->
         select(name='ConcreteSubject')-> notEmpty())
   .class.getAppliedStereotypes()->
      select (name='ConcreteObserver') ->notEmpty ()
```

En esta OCL, se muestra una relación entre participantes en el paquete. Es decir, debe existir una asociación que va desde *ConcreteObserver* a *ConcreteSubject*. Las características dinámicas describen aspectos relacionados al comportamiento de los objetos. Ellas son interacciones que contienen las comunicaciones entre objetos. Los estereotipos que especifican aspectos dinámicos se aplican al diagrama de secuencia UML. Algunas validaciones deben ser realizadas para asegurar que la comunicación entre los objetos es la correcta. Por ejemplo, es importante comprobar la existencia de mensajes y ocurrencias de especificación derivados de operaciones *notify*, que parten y se ejecutan desde líneas de vida estereotipadas como *ConcreteSubject*.

Si el objeto *Subject* cambia su estado, los observadores tienen que ser notificados. La siguiente restricción permite validar la existencia de mensajes que invocan la operación estereotipada *notify.* Y además estos mensajes parten de la línea de vida de un tipo de clase estereotipada como *ConcreteSubject*.

Restricción 2: Invocación de la operación notify.

```
Context PerfilNivel2::ObserverConsistency
Message.allInstances ()-> select(m|m.signature.getAppliedStereotypes()->
   select (name='notify')->notEmpty() and
      m.connector.end.role.type.getAppliedStereotypes()
      ->select (name='ConcreteSubject'-> notEmpty())
```

El tiempo de vida de un objeto *ConcreteObserver* comienza cuando la operación estereotipada *attach* crea una instancia de un tipo de clase estereotipada como *ConcreteObserver*. Si el objeto *ConcreteSubject* cambia su estado, los observadores deben ser notificados. A continuación de la notificación, se actualiza el estado de los objetos de tipo *ConcreteObserver,* al ejecutar la operación *update*.

4 Caso de Estudio

En esta sección se muestra un caso de estudio en el que se aplica el patrón Observer a un modelo. En los diagramas estáticos y dinámicos de UML se puede refactorizar el modelo original aplicando diferentes estereotipos con el fin de especificar un patrón de comportamiento. Al marcar el modelo con estos estereotipos se validan los diagramas de clases y diagramas de secuencia. Se plantea el caso de un sistema de biblioteca,

marcado con el patrón Observer. Las características estructurales se definen con el diagrama de clases UML presentado en la Figura 3 y las características dinámicas con el diagrama de secuencia UML en la Figura 4.

Los estereotipos asociados a los Observadores se aplican en el diagrama de clases UML en clases y métodos. Por ejemplo, la clase *BadStateAlert* está marcada por el estereotipo *Observer* y Administration, Stock, Purchases por *ConcreteObserver*; y finalmente el método refresh por *update*.

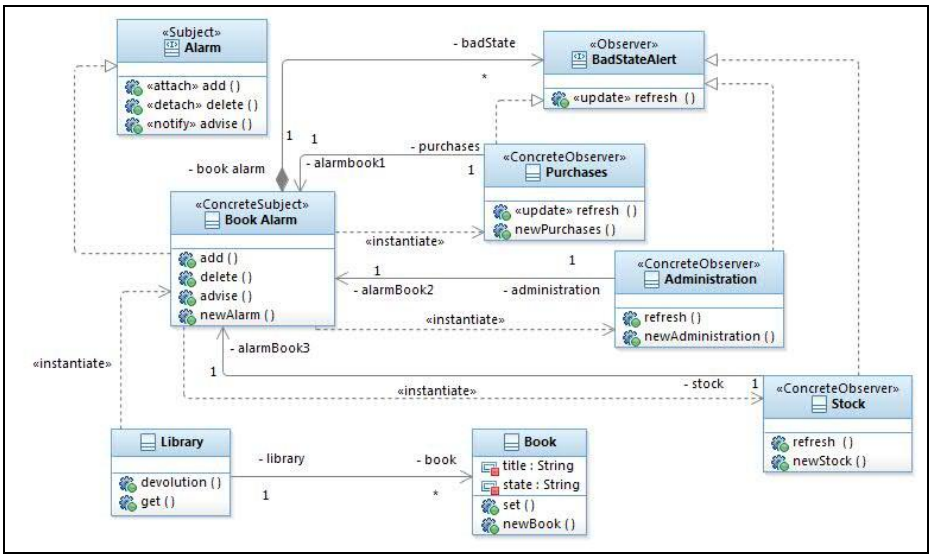

Fig. 3: Diagrama de clases aplicando el patrón Observer.

El diagrama de secuencia UML ilustra un posible escenario de la funcionalidad de devolución de los libros. Cuando un usuario devuelve un libro, la aplicación crea una nueva instancia de la clases Book y Library, Book Alarm (*ConcreteSubject*), Purchase, Administration y Stock (las tres estereotipados con *ConcreteObserver*).

Cuando se crea el libro, se envía un mensaje con el resultado de la operación *get*, que contiene el estado del libro. La aplicación añade los observadores concretos y envía un mensaje para conectar los observadores. Si el estado es malo, entonces el fragmento *Bad State* se ejecuta (acciones recuadradas de la figura 4): comienza la notificación (advise) y se produce la actualización (refresh) del estado del libro.

La consistencia entre los diagramas se garantiza a través del estereotipo ObserverConsistency que representa el paquete que contiene el modelo. Los diagramas de secuencia UML deben ser consecuencia de las definiciones en el diagrama de clases UML. Por ejemplo, la operación refresh (estereotipada con *update*), en el diagrama de clases UML contiene mensajes de invocación en el diagrama de secuencia UML. La aplicación de los estereotipos en diferentes diagramas UML permite establecer si el diagrama especifica correctamente las características del patrón. Pero también permite comprobar la consistencia entre los diagramas del modelo. Por ejemplo, el diagrama de secuencia UML debe incluir instancias de clases y métodos estereotipados en el diagrama de clases UML; tales como, C*oncreteSubject*. Existen entonces, en el diagrama de clases, líneas de vida de un tipo existente en el diagrama de clases estereotipado como C*oncreteSubject.* Lo mismo sucede con la clase estereotipada como *ConcreteObserver*.

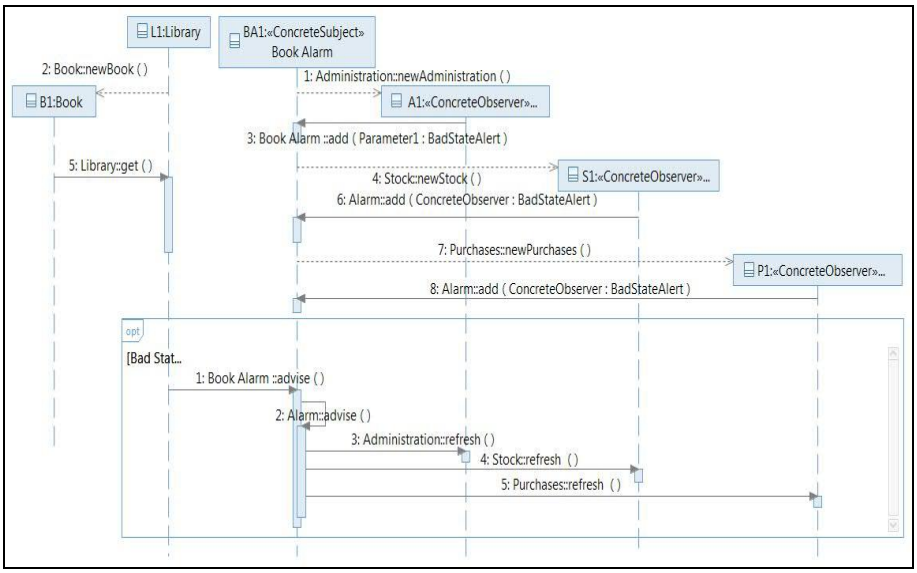

Fig. 4: Diagrama de secuencia aplicando el patrón Observer.

Es posible detectar inconsistencias, a través de la evaluación de restricciones OCL. En este caso particular se utiliza la herramienta RSA, esto se lleva a cabo marcando los modelos con la opción validación. Luego de chequear las restricciones OCL asociadas a los estereotipos, la herramienta informa si los modelos se encuentran bien especificados, de acuerdo a las características del patrón. Por ejemplo, dadas las siguientes restricciones en el contexto *attach y ObserverConsistency,* pertenecientes al Nivel 2.

Restricción 3: Invocación de la operación update.

```
Contexto PerfilNivel2:: ObserverConsistency
    Message.allInstances()->
    select(m|m.signature.getAppliedStereotypes()->
    select(name='update')->notEmpty() and
    m.connector.end.role.type.getAppliedStereotypes()
    ->select (name='ConcreteObserver') -> notEmpty ())
```

Restricción 4: Invocación de la operación attach: argumento.

```
Context PerfilNivel2:: attach
    self.ownedParameter.type.getAppliedStereotypes()->
    select(name='Observer')->size()=1
```

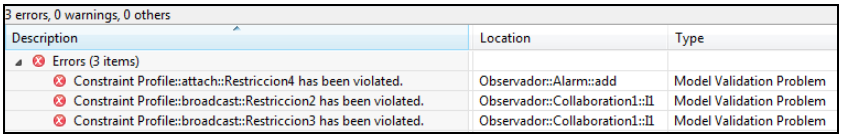

Fig. 5: Validación de restricción en una operación

La figura 5 ilustra la violación de las restricciones. Se violan las restricciones 2 (sección 3) y 3. Al no existir los mensajes de invocación de las operaciones *notify* y *update*, no se cumple el comportamiento. La otra violación corresponde a la falta de un parámetro estereotipado como *Observer*, dentro de la operación *attach,* refiere a un aspecto estructural.

5 Conclusiones y trabajos futuros

Este trabajo ha presentado un enfoque para validar los patrones de diseños de comportamiento de los modelos UML estáticos y dinámicos utilizando los perfiles UML. La propuesta se basa en la implementación de especificaciones a través de OCL, en las tres capas de la arquitectura APPD, la cuál ha sido modelada utilizando una herramienta UML particular, Rational Software Architect.

Las ventajas de la reutilización de buenas prácticas a través del uso adecuado de los patrones de diseño y la modelización de los sistemas utilizando UML, posibilitan a los diseñadores de SW visualizar claramente los patrones y favorecen la comunicación entre ellos; estableciendo un vocabulario común (características propias del uso de patrones de diseño).

El trabajo realizado permitirá avanzar ampliando las definiciones de los patrones de diseño de comportamiento del catálogo Gof, utilizando perfiles de UML y también se pueden incorporar otros tipos de patrones que se alineen a esta metodología.

XMI (por sus siglas en inglés, Exchanging Metadata Information), es un estándar de la OMG, XML Metadata Interchange (2014), que permite compartir modelos UML entre diferentes herramientas de modelado. Rational Software Architect posibilita, exportar también diagramas UML en formato XMI para que después puedan ser importados y modificados en cualquier herramienta que soporte dicho formato.

El uso de XMI permite mediante el intercambio aumentar las funcionalidades: edición de modelos, repositorio de modelos, verificación de modelos y su transformación. Una aplicación interesante y que se proyecta como trabajo futuro es la creación de una transformación que sea un puente hacia herramientas OCL. Verificando aspectos dinámicos como pre y postcondiciones de un modelo y/o herramientas de testeo de software. De manera de asegurar mayor cantidad de chequeos de los modelos UML. Y generar un repositorio de transformaciones para aspectos como generación de código.

La especificación completa del presente trabajo se encuentra disponible en: www.um.edu.ar/es/fi/informacion/institutos-facultad-de-ingenieria/instituto-informatica/publicaciones.html

Referencias

Gamma, E., Helm, R., Johnson, R., Vlissides J. (1994) “Design Patterns: Elements of Reusable software”, Addison-Wesley.

Taibi, T. (2007) Design Patterns Formalization Techniques, pp. 1--206 IGI Publishing.

Lagarde, F., Espinoza, H., Terrier, F., Gérard, S. (2007) “Improving UML profile design practices by leveraging conceptual domain models”, 22th IEEE/ACM International Conference on Automated Software Engineering, p. 445–448.

Debnath, N., Garis, A., Riesco, D., Montejano, G. (2006) “Defining Patterns using UML Profiles”. ACS/IEEE International Conference on Computer Systems and Applications, IEEE Press, www.ieee.org, pp.1147-1150.

Dye, S. (2006) “Formal specification of structural and behavioral aspects of design patterns”. Journal of Object Technology, Volume 9, no. 6 (November 2010), pp. 99-126, 2010.

Flores, A., Cecchi, A., Aranda, G. (2006) A Generic Model of Object-Oriented Patterns Specified in RSL. XXXII Latin American Informatics Conference (CLEI).

Dae–Kyoo, K., Wuwei, S. (2007) “An approach to evaluating structural pattern conformance of UML models”. Proceedings of the 2007 ACM symposium on Applied Computing, pp. 1404 – 1408.

Cortez, A., Garis, A., Riesco, D. (2012) “Perfiles UML para la especificación de patrones de comportamiento.Un caso de estudio”, CACIC -XVIII Congreso Argentino de Ciencias de la Computación. Bahía Blanca, SF, Argentina.

Cortez, A., Garis, A. (2012) “Aplicación de Perfiles UML en la especificación de Patrones de comportamiento”, 41JAIIO -SADIO. La Plata, SF, Argentina.

Rational Software Architect. (2008), Link http://www-306.ibm.com/software/rational.

Meyer, B., Arnout, K. (2006) “Componentization: The Visitor Example”, to appear in Computer (IEEE).

OMG: UML Superstructure, (2011) version 2.4.1.

OMG: Object Constraint Language, (2014) version 2.4.

Pabitha P., Shobana Priya A.: Rajaram M. (2012) “An Approach for Detecting and Resolving Inconsistency using DL Rules for OWL. Generation from UML Models” ISSN 1450-216X T.

OMG: XML Metadata Interchange (XMI), (2014) version 2.4.2.