Perfiles UML para la Especificación de Patrones de Comportamiento: Un Caso de Estudio

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

[1] Consejo de Investigaciones, Universidad del Aconcagua, Mendoza, Argentina.
[2] Instituto de Informática, Universidad de Mendoza, Mendoza, Argentina.
cortezalberto@gmail.com
[3] Universidad Nacional de San Luis, San Luis, Argentina
{agaris, driesco}@unsl.edu.ar

Resumen. Los Patrones de Diseño, como técnica de Ingeniería de Software, mejoran la calidad en el proceso de desarrollo. Los Patrones de Diseño de Comportamiento (según la clasificación GoF) definen aspectos dinámicos del sistema, entre ellos uno de los aspectos más complejos como lo es el flujo de control. De esta manera los patrones facilitan el modelado de sistemas, reduciendo paralelamente tiempo y costos. A este punto es substancial la eliminación de ambigüedades en su representación, para poder aplicarlos en el proceso de modelado. En consecuencia surge la necesidad de elaborar especificaciones precisas que posibiliten su aplicación y validación. El presente trabajo muestra un caso de estudio en el que se aplica a un ejemplo un Patrón de Comportamiento, especificado a través de los Perfiles UML y el lenguaje OCL. Dichas técnicas conforman un procedimiento para la especificación y validación de Patrones de Diseño de Comportamiento.

Palabras clave: Patrones de Diseño, Perfiles UML, OCL.

1. Introducción

Los beneficios que provee el uso de patrones de diseño en el proceso de desarrollo de software incluyen: reutilización de diseño y del código potencial, mayor comprensión de la organización global de un sistema, y mejor interoperabilidad con otros sistemas mediante la introducción de estándares [2, 13]. Gamma y otros en [2] presentan 23 patrones de diseño, clasificados según dos criterios: el propósito y el ámbito. Según el propósito se catalogan en patrones de creación (crean objetos), estructurales (componen estructuras) y de comportamiento (interacción de los objetos). Pero el uso de estos patrones se ve dificultado por las ambigüedades que surgen a la hora de su implementación. Los ejemplos de código y los diagramas mostrados en el catalogo pertenecen a ejemplos específicos y no se pueden aplicar como regla general. Así surge la necesidad de formular una especificación estándar que descarte interpretaciones erróneas.

Una notación estándar para aplicar un patrón y facilitar su aplicación y validación fue investigada en [5,6, 8, 9, 10, 15,18, 19, 20]. Se utilizó la lógica de primer orden y la lógica temporal para especificar tanto aspectos estructurales como dinámicos [5,

18], así como también la notación semántica *calculus* [19, 20]. Se empleo algunos lenguajes formales como RAISE [9], asi como también herramientas para verificación lógica como SPIN o la tecnología de web semántica [6, 8]. Pero cada uno de estos enfoques no tuvo un uso extendido entre los especialistas de software por dos razones. La primera es que la mayoría de los trabajos fueron realizados en base a complejas notaciones matemáticas en lugar de utilizar modelos simples. Y la segunda es que solo los estudios más complejos describieron los aspectos dinámicos.

El lenguaje de modelado UML es un estándar utilizado para especificar y documentar sistemas [7]. Los perfiles UML son la herramienta UML que extiende su sintaxis y su semántica, de manera de expresar los conceptos de un determinado dominio de aplicación. Los perfiles UML han sido propuestos anteriormente para la especificación de patrones de diseño en [1, 3]. Sin embargo, dichos enfoques están limitados solo a patrones del tipo estructural, dejando de lado la posibilidad de representar atributos en diagramas UML dinámicos, tales como diagramas de secuencia.

El presente trabajo propone los perfiles UML como mecanismo para especificar patrones de diseño de comportamiento. Se presenta un caso de estudio representado a través de diagramas de clase y de secuencia. A dichos diagramas se les aplican perfiles que contienen las especificaciones de un patrón de comportamiento. Las especificaciones están construidas a través de estereotipos y restricciones OCL. Se validan verificando los estereotipos aplicados y el cumplimiento de las restricciones formuladas. Con este procedimiento se definen los patrones en dos de los diagramas más populares de UML; utilizando herramientas UML existentes sin tener que definir nuevas. En particular, se emplea la el contexto de la herramienta Rational Software Architect [11, 12], la cual permite definir perfiles UML y validar especificaciones presentadas en OCL (adoptado dentro del perfil UML expuesto) [8, 13, 14, 17]. Este trabajo propone un instrumento que integre diagramas estructurales con diagramas de comportamiento para ser usados con los modelos UML.

La organización del trabajo es la siguiente: en la Sección 2 se presentan los conceptos base para la comprensión del trabajo: perfiles UML y patrones de comportamiento. En la Sección 3 se explica el enfoque, presentando la especificación de del patrón de diseño Cadena de Responsabilidad [2]. En la sección 4 se presenta un ejemplo de su empleo. En la Sección 5 se exponen las conclusiones y líneas de trabajo futuro.

2. Conceptos Preliminares

Esta sección incluye nociones preliminares. La Sección 2.1 explica conceptos asociados a patrones de diseño de comportamiento y la Sección 2.2 detalla los perfiles UML.

2.1. Patrones de diseño de comportamiento

Para diseñar un sistema que esté preparado para los cambios, se debe considerar como el sistema podría cambiar a lo largo de su ciclo de vida. Un diseño que no tiene en cuenta el cambio se arriesga a no poder ser rediseñado en el futuro. Esos cambios podrían involucrar tanto modificación y reimplementación de clases como retesteo.

Los patrones del diseño ayudan al rediseño, asegurando que un sistema puede cambiar en las formas específicas. Cada patrón del diseño se ocupa de algún aspecto del sistema para una clase particular de cambio. Existen diversas causas de rediseño como los tipos de dependencia de operaciones, algoritmos, implementaciones, hardware y software. Existen dos conceptos aplicados por los patrones de diseño: *composición* y *delegación*. Composición es una alternativa a la herencia de clases y se obtiene ensamblando objetos para obtener más funcionalidad. La composición disminuye la dependencia de implementación. Un diseño apoyado en la composición de objetos tendrá más objetos y el comportamiento del sistema estará condicionado por sus interrelaciones en lugar de estar definido en una clase. Delegación es una poderosa forma de composición para la reutilización de software. En la delegación, dos objetos manejan una petición: un objeto receptor delega operaciones a su delegado**.** Los patrones de comportamiento se ocupan de los algoritmos y la asignación de responsabilidades en los objetos. Se enfocan en la comunicación entre objetos. Según sea el caso los patrones de diseño usan la herencia para distribuir comportamiento entre clases, usan composición de objetos por sobre la herencia.

2.2. Perfiles UML

El perfil es un mecanismo definido por UML para extender y adaptar UML a una plataforma o dominio particular [7]. Un perfil UML se define como un conjunto de estereotipos, restricciones y valores etiquetados. A través de los ***estereotipos*** se pueden crear nuevos tipos de elementos a partir de elementos que ya existen en el metamodelo UML. Los estereotipos están definidos por un nombre y algunos elementos del metamodelo a los que puede asociarse. Se representa gráficamente con su nombre entre paréntesis angulares << nombre-estereotipo >>. Las ***restricciones*** imponen condiciones que deben cumplir algunos o varios elementos del modelo para que esté "bien formado", según un dominio de aplicación específico. Una restricción puede ser representada como una cadena de caracteres entre llaves colocadas junto al elemento al que está asociada o conectada a él por una relación de dependencia. Es posible definir una restricción mediante una expresión OCL. Un ***valor etiquetado*** es una extensión de las propiedades de un elemento de UML permitiendo añadir nueva información en la especificación del elemento. Se representa como una cadena de caracteres entre llaves asociada al nombre del elemento. La cadena incluye un nombre (etiqueta), un separador (=), y un valor (el de la etiqueta). Para obtener un perfil se tiene que especializar un subconjunto de UML a través de estereotipos, restricciones y valores etiquetados.

Fuentes et al. en [4] sugieren una metodología para la definición de un Perfil UML. Dicha metodología, base para definir perfiles de patrones de comportamiento en el presente trabajo, es descripta a continuación.

(1) Antes de comenzar, es preciso disponer de la correspondiente definición del metamodelo de la plataforma o dominio de aplicación a modelar con un Perfil. Si no

existiese, entonces se necesita definir dicho metamodelo utilizando los mecanismos del propio UML (clases, relaciones de herencia, asociaciones, etc.), de la forma usual como se realizaría si nuestro objetivo no fuese definir un perfil UML. Incluir la definición de las entidades propias del dominio, las relaciones entre ellas, así como las restricciones que limitan el uso de estas entidades y de sus relaciones.
(2) Si se dispone del metamodelo del dominio, pasar a definir el perfil. Dentro del paquete «profile» se incluye un estereotipo por cada uno de los elementos del metamodelo que deseamos incluir en el perfil. Estos estereotipos tendrán el mismo nombre que los elementos del metamodelo, estableciéndose de esta forma una relación entre el metamodelo y el Perfil. En principio cualquier elemento que hubiésemos necesitado para definir el metamodelo puede ser etiquetado posteriormente con un estereotipo.
(3) Es importante tener claro cuáles son los elementos del metamodelo de UML que se está extendiendo sobre los que es posible aplicar un estereotipo. Ejemplo de tales elementos son las clases, sus asociaciones, sus atributos, las operaciones, las transiciones, los paquetes, etc. De esta forma cada estereotipo se aplicará a la metaclase de UML que se utilizó en el metamodelo del dominio para definir un concepto o una relación.
(4) Definir como valores etiquetados de los elementos del Perfil los atributos que aparezcan en el metamodelo. Incluir la definición de sus tipos, y sus posibles valores iniciales.
(5) Definir las restricciones que forman parte del Perfil, a partir de las restricciones del dominio. Por ejemplo, las multiplicidades de las asociaciones que aparecen en el metamodelo del dominio, o las propias reglas de negocio de la aplicación deben traducirse en la definición las correspondientes restricciones.

3. Perfil UML para el Patrón de Diseño Cadena de Responsabilidad

En esta sección se presenta la especificación de un patrón de diseño de comportamiento particular, Cadena de responsabilidad (en inglés Chain of Responsability), mediante el uso de perfiles UML y su modelado utilizando RSA. La idea presentada consiste en representar mediante un perfil UML tanto las características estructurales como dinámicas del patrón. Además se eliminan ambigüedades estableciendo restricciones dentro de los estereotipos del perfil. A continuación se describen las características del patrón y como se lleva a cabo su especificación.

El patrón de diseño denominado *Chain of Responsability* permite evitar el acoplamiento entre el receptor y el emisor de un mensaje, construyendo una cadena de objetos receptores. De esta manera cuando se envía una petición a más de un objeto, cualquiera de ellos puede responder la solicitud. Los principales elementos del patrón, según el catálogo GoF, son *Handler*, *ConcreteHandler, HandlerRequest* y *successor*. *Handler* es una interface que especifica las peticiones del cliente. Dicha petición, denominada genéricamente como *HandlerRequest*, también es participante del patrón. El elemento *ConcreteHandler* implementa *HandlerRequest*. El elemento *successor* simboliza el próximo integrante de la cadena. Asi queda definido el perfil

del patrón *Chain of responsability* de la Fig. 1. Por cada participante principal del patrón se especifica un estereotipo, al que se le añaden restricciones OCL para establecer las características propias del patrón.

Las características estructurales y de comportamiento se definieron respectivamente a través de un perfil estructural y un perfil de comportamiento. El Perfil Estructural contiene los estereotipos *TipoClasificador* y *TipoClase* (Fig. 1). Se recurre a TipoClasificador para definir el estereotipo *Handler* como una interface. TipoClase proporciona las características de una clase concreta que son asignadas a *ConcreteHandler*. Y los estereotipos restantes *HandlerRequest* y *successor* extienden respectivamente las metaclases de UML *Operation* y *Association*.

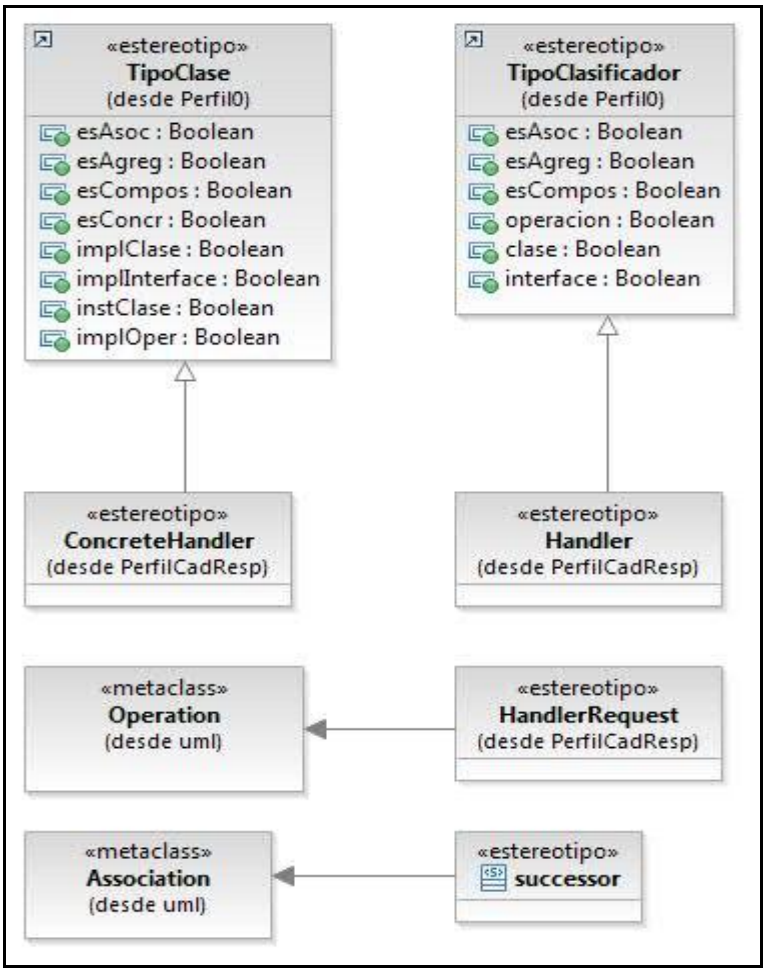

Fig. 1. Perfil de Patron Cadena de Responsabilidad

El perfil de comportamiento se compone de dos estereotipos aplicados directamente al paquete: *Elemento* y *Comunicador* (Fig. 2). El estereotipo *Elemento* representa los elementos del diagrama de secuencia, instancias en tiempo de ejecución del

diagrama de clases. Con este estereotipo se valida la consistencia entre el diagrama de clases y el diagrama de secuencia. El estereotipo *Comunicador* representa los patrones de comunicación entre objetos y permite validar las interacciones dentro de un patrón. Este perfil se aplica a un patrón de diseño de comportamiento particular para precisar sus características dinámicas.

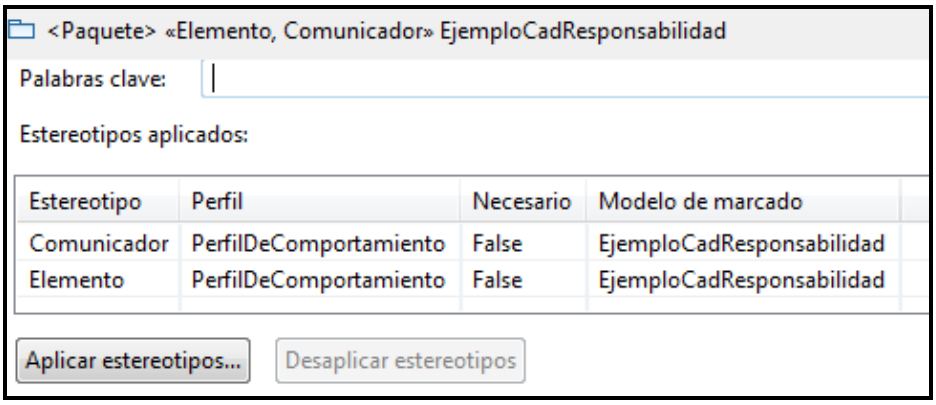

Fig. 2. Estereotipos del Perfil de Comportamiento

Los atributos permiten activar o desactivar las características de un determinado patrón según corresponda. Los atributos son activados si la restricción es aplicable al patrón particular tratado. En otro caso se encuentran desactivados. Por ejemplo, el atributo *esAgreg*, asociado al estereotipo *TipoClasificador*, define que la clase contiene una agregación. El atributo *esAgreg* está activado si su valor es *true*. La restricción OCL correspondiente a esta validación se muestra a continuación.

```
context DesignPatternFrameworkProfile::TipoClasificador
inv:  self.esAgreg  implies  self.attribute->
select(a|a.aggregation=uml::AggregationKind::shared)->notEmpty()
```

En el caso del perfil de comportamiento también se trabaja con atributos. Por ejemplo el estereotipo *Elemento* contiene el atributo de tipo booleano que permiten definir si el patrón crea objetos: *creaInst*. Para verificar que exista un mensaje de creación en el diagrama de interacción se formula la especificación OCL:

```
context BehaviorProfile::Elemento
inv: createMessage.Interaction.allInstances().message->
exists(m|m.messageSort=uml::MessageSort::createMessage)
```

Se puede visualizar mejor la aplicación de los conceptos a través de un caso de estudio en el que se aplica el perfil del patrón *Chain of Responsability*.

4. Aplicación Perfil UML a un Caso de Estudio

Sea el caso de una función del sistema, que detecta un evento del que debe informar mediante un mensaje. Es necesario desacoplar al emisor (el sistema), de los posibles receptores (que generan los denominados logs). El primer objeto de la cadena recibe la petición y, o bien la procesa, o bien la envía al siguiente objeto de la cadena, que hará exactamente lo mismo. Con la aplicación del perfil se desacopla al emisor (en este caso el registro) que informa, de los posibles receptores (los encargados del registro de los eventos). En este caso el emisor es *ManejaRegistrodeErrores* y los receptores serán: RegistraErroresDebug, RegistraErroresEmail y RegistraErroresStandard que se muestran en la Fig. 3.

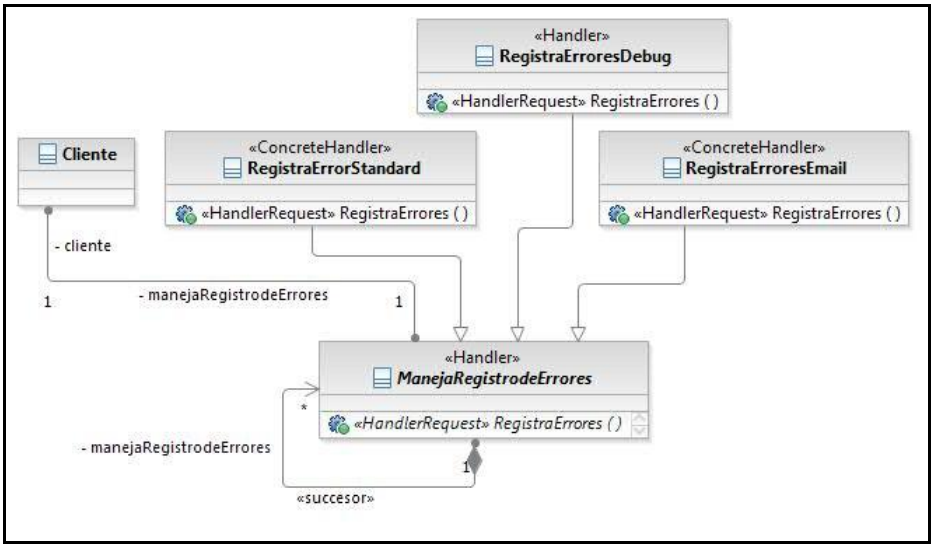

Fig. 3. Diagrama de clases generador de logs

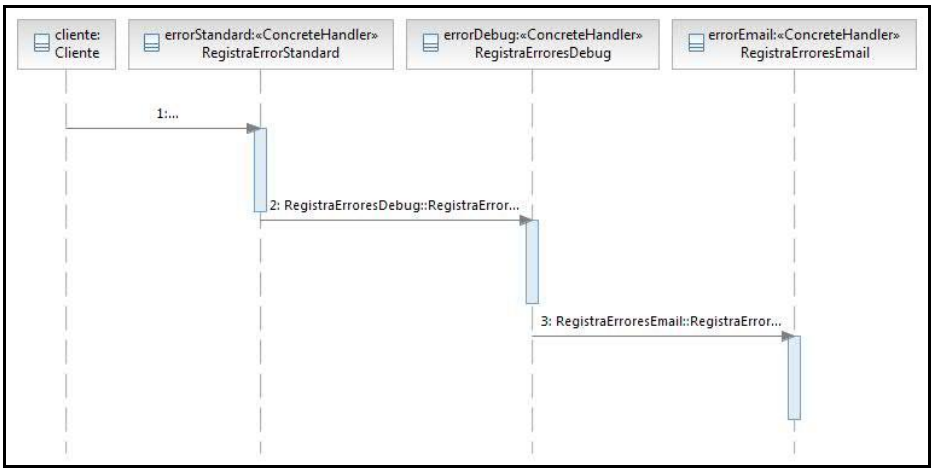

Fig. 4. Ejemplo 1 Diagrama de secuencia generador de logs

En la Fig. 4 se presenta el diagrama de secuencia para este caso de estudio. En la Fig. 5 se muestra otro diagrama de secuencia que también utiliza los estereotipos definidos en el perfil Cadena de Responsabilidad. Si se toma el diagrama de clases de la Fig. 3 al validar el modelo se generan errores como se muestra en la Fig. 6. El primer error "Se ha violado la restricción Perfil::Elemento::ConsistenciaLifelines" se refiere a que existen lifelines no se corresponden con las clase existentes. El segundo error "Se ha violado la restricción Perfil::Elemento::ConsistenciaMensajes " cuando no existe la consistencia entre mensajes y asociaciones.

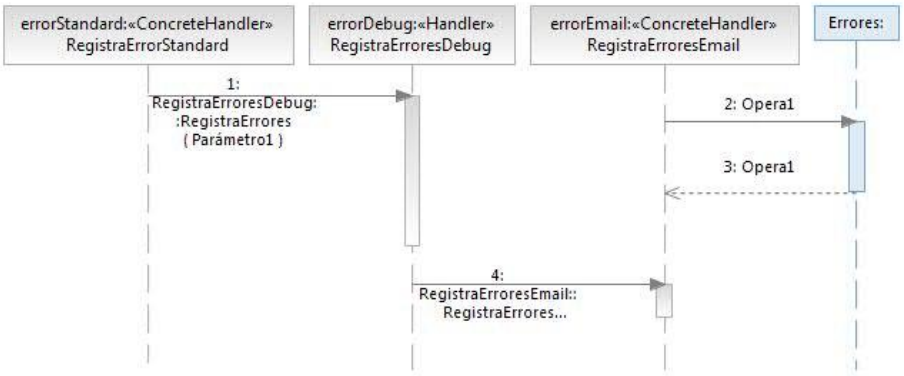

Fig. 5. Ejemplo 2 Diagrama de secuencia generador de logs

El error "Se ha violado la restricción Perfil::Elemento::ConsistenciaOperaciones" se produce cuando existen mensajes que no se corresponden con una operación de clase. Es decir, se comprueba la consistencia entre los diagramas y se verifica que el diagrama de secuencia refleje las colaboraciones especificadas para el patrón de diseño.

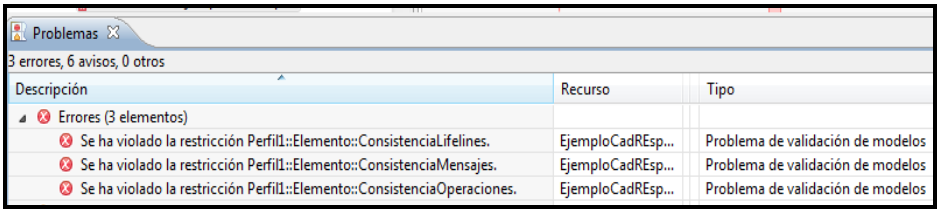

Fig. 6. Ejemplo de errores generados por la validación

La especificación completa del caso de estudio presentado, está disponible para bajar en ***https://sites.google.com/site/proyectociuda.***

5. Conclusiones

El presente trabajo ha mostrado la formalización de un patrón de diseño de comportamiento, *Chain of Responsability*, a través de perfiles UML. Dicha formalización ha sido modelada utilizando una herramienta UML particular, Rational Software Architect. El patrón es descripto a través de estereotipos que contienen atributos representando sus características estructurales y de comportamiento. Respecto de las características de comportamiento se muestran las cualidades de consistencia y de comunicación. De esta manera, es posible verificar las interacciones existentes entre objetos, formalizar los esquemas de comunicación entre objetos y validar condiciones de consistencia entre los diagramas de clase y de secuencia.De esta manera se mejora la calidad de los modelos al verificar si se corresponden con un patrón de comportamiento determinado.

Al mismo tiempo, un modelo UML es más preciso si contiene restricciones OCL, a través de lo que se denomina una transformación de OCL[1]. Por lo cual se puede aplicar a los modelos los estereotipos definidos en este trabajo, para transformarlos en modelos más minuciosos y rigurosos.

Como corolario el trabajo analizado se puede emplear como base para transformaciones con el IDE Eclipse o generando el formato de intercambio XMI para su uso con otros IDE.

Como trabajo futuro, se intenta avanzar en el chequeo de inconsistencias de aspectos más complejos, adicionar nuevos patrones de diseño, así como enriquecer los perfiles para permitir incorporar estereotipos a otros diagramas UML, tales como diagramas de estado.

[1] Una transformación de OCL hace corresponder uno o más elementos de un modelo *fuente* en uno o más elementos de un modelo *destino* [16].

Referencias

1. Debnath N., Garis A., Riesco D., Montejano G.: Defining Patterns using UML Profiles. *ACS/IEEE International Conference on Computer Systems and Applications*, IEEE Press, www.ieee.org, pp.1147-1150 (2006)
2. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns: Elements of Reusable software, Addison-Wesley, pp. 205-317 (1994)
3. Garis, A.: Perfiles UML para la definición de Patrones de Diseño. Tesis de maestría. Universidad Nacional de San Luis (2007)
4. Fuentes, L., Vallecillo A., Troya, J.: Using UML Profiles for Documenting Web-Based Application Frameworks, Annals of Software Engineering , 13:249-264. (2002)
5. Dae –Kyoo, K., Wuwei, S.: An approach to evaluating structural pattern conformance of UML models. Proceedings of the 2007 ACM symposium on Applied computing, pp. 1404 – 1408 (2007)
6. Dietrich, J., & Elgar, C.: Towards a Web of patterns. Journal of Web Semantics, Volume 5 , Issue 2 (June), pp 108-116 (2007)
7. OMG: UML Superstructure, Version 2.4.1 (2011)
8. Blewitt, A.: Spine: Language for Pattern Verification. Design *Pattern* Formalization Techniques. IGI Global, pp. 109 -122 (2007)
9. Flores, A., Cecchi, A., Aranda, G.: A Generic Model of Object-Oriented Patterns Specified in RSL. XXXII Conferencia Latinoamericana de Informática (CLEI) (2006)
10. Taibi, T.: Design Patterns Formalization Techniques, pp. 1-44 IGI Publishing. (2007)
11. Misic, D.: Authoring UML Profiles: Using Rational Software Architect, Rational Systems Developer, and Rational Software Modeler to create and deploy UML Profiles. Disponible en http://www.ibm.com/developerworks/rational/library/08/0429_misic1/ (2008)
12. Rational Software Architect. Disponible en http://www-306.ibm.com/software/rational (2008)
13. Bhutto, A.: Formal Verification of UML. Australian Journal of Basic and Applied Sciences, 5(6): 1594-1598 (2011)
14. Pabitha, P., Shobana Priya, A., Rajaram, M.: An Approach for Detecting and Resolving Inconsistency using DL Rules for OWL Generation from UML Models. ISSN 1450-216X Vol.72 No.3 (2012), pp. 404-413 European Journals of Scientific Research Publishing, Inc (2012)
15. Pavlic, L. Hericko, M. Podgorelec, V.: Improving Design Pattern Adoption with an Ontology-Based Repository. Conferencia Information Technology Interfaces, ITI 2008. 30th International, pp. 649 – 654 (2008)
16. OMG: MDA Guide versión 1.0.1 (2003)
17. OMG: Object Constraint Language, Version 2.3.1 (2012)
18. Taibi,T., Mkadmi,T.: Formal specification of Design patterns-A balanced approach.*Journal of Object Technology, 2*(4), 127-140 (2003)
19. Smith, J.: SPQR: Formal foundations and practical support for the automated detection of Design patterns from source code. Doctoral thesis, University of North Carolina at Chapel Hill, pp. 23-26 (2005)
20. Smith, J.McC., & Stotts, D. Elemental Design patterns: A formal semantics for composition of OO software architecture. In Proceedings of the 27th Annual IEEE/NASA Software Engineering Workshop (pp. 183-190). (2002)