Aplicación de Perfiles UML en la Especificación de Patrones de Comportamiento

Alberto Cortez[1 2] y Ana Garis[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@unsl.edu.ar

Resumen. Los Patrones de Diseño como herramienta de la Ingeniería de Software, proponen soluciones a problemas recurrentes en el desarrollo de software. Los Patrones de Diseño de Comportamiento (según la clasificación Gof) permiten conocer no solo aspectos deseables del sistema de índole estructural sino especialmente sus características dinámicas. El uso de Patrones de Comportamiento requiere de una especificación precisa que facilite tanto su definición, como también su aplicación y validación. En éste sentido, el presente trabajo propone el uso de Perfiles UML y OCL para alcanzar éste propósito. El enfoque habilita la aplicación de Perfiles UML como mecanismo para la especificación y validación de los patrones, tanto en modelos estáticos como en modelos dinámicos del sistema.

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

1. Introducción

Los Patrones de Diseño describen problemas recurrentes en el diseño de software y su solución, utilizando técnicas que han demostrado su utilidad a lo largo de muchos años. Los beneficios que provee el uso de patrones 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. 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). La búsqueda de una notación capaz de modelar la esencia de un patrón y facilitar su definición, aplicación y validación ha sido emprendida por diversos autores [6, 11, 16]. Los principales escollos encontrados en estos enfoques, han sido la falta de estandarización, por ende la necesidad de crear herramientas específicas y el uso de metodologías que no son de uso extendido.

El lenguaje de modelado UML es un estándar utilizado para especificar y documentar sistemas [8]. Los Perfiles UML son proporcionados por el propio UML para

extender su sintaxis y su semántica, de manera de expresar los conceptos de un determinado dominio de aplicación. Un perfil UML normalmente incluye un conjunto de estereotipos, restricciones y valores etiquetados. Los primeros permiten crear nuevos elementos, a partir de otros que ya existen en el metamodelo UML; las restricciones establecen condiciones a cumplir por los estereotipos, y los valores etiquetados habilitan la adición de información extra.

Los perfiles UML han sido propuestos para la especificación de patrones de diseño e incorporados en una *Arquitectura de Perfiles UML de Patrones de Diseño* (APPD) en [1, 4]. APPD 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 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 UML de comportamiento del 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 [8].

El presente trabajo propone valerse de perfiles UML como mecanismo para especificar Patrones de Diseño de Comportamiento (PDC). Con este procedimiento se puede aplicar los PDC en los modelos de sistemas; concretamente, en diagramas de clase y diagramas de secuencia UML [8]. El enfoque permite validar las especificaciones expresadas en los perfiles UML, para un conjunto de diagramas de un modelo. Para que esto sea posible es necesario enriquecer APPD en cada uno de los niveles, especialmente el perfil UML de patrones de comportamiento (en el Nivel 1) y los perfiles individuales por cada uno de los PDC en el Nivel 2. Las características requeridas de cada perfil son precisadas estableciendo estereotipos y asociándole restricciones OCL. Así, para definir un patrón particular en el Nivel 2, se incluye un estereotipo por cada elemento participante del patrón y se le añaden las restricciones OCL correspondientes. Por ejemplo, para modelar el Patrón Command [2], se identifican como elementos participantes a *Invoker*, *Receiver*, *Command* y *ConcreteCommand*, de manera que se define un estereotipo por cada uno de los participantes y se imponen en OCL las características propias de cada elemento.

El trabajo aquí presentado contribuye en la especificación no sólo de aspectos estructurales relacionados a PDC, sino también de comportamiento; de ésta forma éstos pueden ser utilizados tanto en diagramas de clases UML como en diagramas de secuencia UML. Esto posibilita la verificación de consistencias entre diagramas y la visualización de los diferentes elementos del patrón relacionados entre sí; añadiendo los estereotipos que definen los elementos de cada patrón en los dos diagramas estático y dinámico antes mencionados. Es importante mencionar, que los beneficios que genera la definición de patrones de diseño con perfiles UML [1, 4], se aplican en particular a los PDC. Entre algunos de los puntos más relevantes es posible mencionar que habilita a los ingenieros de software a incorporar patrones en dos de los diagramas más populares de UML, como son los diagramas de clase y diagramas de secuencia; al tiempo que permite utilizar herramientas UML existentes sin tener que definir

nuevas. En particular, la APPD, presentada en [1, 4], junto con las nuevas especificaciones de PDC, son modeladas, en el contexto de la herramienta Rational Software Architect [13], la cual permite definir perfiles UML y validar especificaciones OCL. Cabe señalar que la propuesta se alinea con el enfoque MDA [17, 10] (por sus siglas en inglés de Model Driven Architecture), dando lugar a que los modelos marcados puedan ser luego transformados a otros modelos o a código.

La organización del trabajo es la siguiente: en la Sección 2 se reseñan trabajos relacionados, realizando un análisis de las propuestas. En la Sección 3, se presenta un resumen de los conceptos necesarios para comprender el trabajo: perfiles UML y arquitectura de patrones de diseño. En la Sección 4 se detalla el trabajo realizado, a saber: la arquitectura construida con la definición de los correspondientes perfiles UML, las especificaciones desarrolladas, y su aplicación a un ejemplo concreto. Para finalizar, en la Sección 5 se exponen las conclusiones y se presentan las líneas de trabajo futuro.

2. Trabajos Relacionados

A continuación se describen algunos trabajos de investigación, en donde se presentan diferentes enfoques para la especificación de PDC.

Dae-Kyoo Kim, France, Ghosh y Eunjee Song expusieron en [11] acerca del RBML, un lenguaje de especificación de patrones, capaz de capturar las propiedades estructurales y de comportamiento. Las especificaciones son definidas usando roles. El lenguaje ha sido usado para especificar los patrones: *Abstract Factory*, *Bridge*, *Decorator*, *Singleton*, *Observer*, *Visitor*, *Iterator and Composite*. El RBML fue desarrollado como complemento de la herramienta Rational Rose, es decir, es una extensión de sólo una herramienta de UML.

Pavlič et al. [16] presentaron la plataforma OBDPR (por sus siglas en inglés, Ontology-Based Design Pattern Repository), un repositorio de patrones de diseño automáticos e inteligentes. El objetivo principal de OBDPR es introducir métodos formales en la representación de patrones de diseño con el fin de aprovechar las conocidas capacidades de la inteligencia artificial. Es una excelente recopilación del uso de los métodos formales en la especificación de patrones pero, carece de aceptación entre los ingenieros de software familiarizados con UML, que normalmente no utilizan métodos formales.

Barotto et al. en [7] proponen la definición de patrones modificando el metamodelo UML. Se agregó una metaclase denominada *Pattern* para representar los patrones y las metarelaciones *formadopor* y *tiene*. La primera determina los componentes de un patrón y la segunda las restricciones OCL para su especificación y validación. El trabajo fue implementado en ArgoUML, creándose para ello estereotipos y restricciones codificadas en Java. En esta adaptación se describe un conjunto reducido de patrones: *Singleton, Template Method, Decorator, Composite y Chain of Responsability*. Se usa la versión 1.3 de UML, en consecuencia no se pudo aplicar el concepto de perfil para la extensión del metamodelo, dado que dicho concepto fue incorporado en

la versión 2 de UML. De esa manera no se generó un concepto estandarizado, sino uno particular.

El lenguaje de especificación del patrón equilibrado en [11] es otro desarrollo para la estructura y el comportamiento de patrones de diseño en tres niveles de abstracción: la composición de patrones, los patrones, y las instancias de patrones. Se utiliza la lógica de primer orden para especificar el aspecto estructural de los patrones, mostrando las relaciones entre los participantes como predicados. Se recurre a la lógica temporal para especificar el comportamiento de patrones, ya que es la más adecuada para describir el comportamiento colectivo de objetos. La utilización de formalismos diferentes para los aspectos estructurales y de comportamiento plantea ciertos inconvenientes de índole práctica. Estas dificultades se presentan al no existir una herramienta estándar para implementar ambas especificaciones en un mismo modelo.

N. Debnath et al. en [1] analizan las ventajas del uso de perfiles para definir, documentar y visualizar patrones de diseño. Se propone una arquitectura de niveles usando perfiles UML. En el trabajo de Garis [4] se define la APPD para la especificación de patrones y se detallan algunos patrones estructurales (según la clasificación Gof). Aunque en [1, 4] se define una arquitectura para perfiles de patrones y se especifican algunos patrones estructurales, la arquitectura no fue completamente implementada en una herramienta UML particular, por lo que dicha propuesta fue testeada parcialmente. Por otro lado, los trabajos mencionados, especifican únicamente características estructurales de los patrones de diseño, por lo que no es posible; por ejemplo, utilizar perfiles de patrones sobre diagramas dinámicos, tales como diagramas de secuencia.

3. Conceptos Preliminares

Esta sección contiene una sinopsis de los conceptos base empleados para la formulación del presente trabajo. En la Sección 3.1 se expone el mecanismo de perfiles UML y el lenguaje de restricciones OCL. En la Sección 3.2 se explica la estructura de la arquitectura aplicada.

3.1. Perfiles UML

El perfil es un mecanismo definido por UML para extender y adaptar UML a una plataforma o dominio particular [8]. 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.

Un diagrama UML, por ejemplo un diagrama de clases, no está lo suficientemente refinado para proveer todos los aspectos relevantes de una especificación. Existe, entre otras cosas, una necesidad de describir restricciones adicionales sobre objetos del modelo. OCL [18] es un lenguaje de especificación, el cual garantiza estar libre de efectos laterales. Cuando se evalúa una expresión OCL, simplemente retorna un valor, sin hacer modificaciones en el modelo. OCL no es un lenguaje de programación, es decir, no es posible escribir la lógica de un programa o flujos de control.

3.2. Arquitectura de Patrones de Diseño

Los perfiles UML fueron propuestos para la definición de patrones de diseño estructurales en [1, 4]. Fue creada una arquitectura para la reutilización de los Perfiles UML, siendo éste un paso clave en el diseño de PDC. En la Figura 1 se puede observar la estructura de la arquitectura mencionada anteriormente. El Nivel 0, se compone del DPFP (por sus siglas en inglés de, Design Pattern Framework Profile), que contiene las características comunes para todos los perfiles de patrones. En este nivel se definen *estereotipos*, *valores etiquetados* y *restricciones* generales.

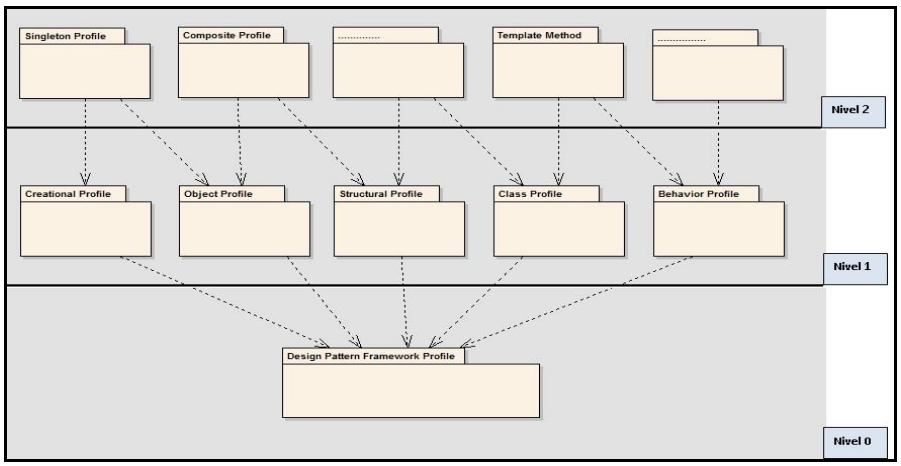

Fig. 1. Arquitectura para patrones usando perfiles UML

El Nivel 1 se inspira en el catálogo Gof, y contiene la clasificación por propósito dividida en estructurales (Structural Profile), creacionales (Creational Profile), y de comportamiento (Behavior Profile); y la clasificación en perfiles de clase (Class Profile) u objeto (Object Class). El Nivel 2 es el nivel superior de este modelo, está com-

puesto por los perfiles de patrones propiamente dichos, con los perfiles individuales elaborados por cada patrón. Cada uno de ellos podrá usar los elementos definidos en los niveles inferiores (niveles 0 y 1). Cuando se incluye un nuevo patrón, se debe definir un nuevo perfil, y cada perfil agregado tendrá su sintaxis y semántica particular. Pero todos absorberán la estructura genérica del nivel anterior. Un perfil de patrón importará al menos dos perfiles del nivel inferior. Los niveles 0 y 1 de la arquitectura están compuestos por un conjunto de perfiles que pueden ser aplicados en uno o varios perfiles o modelos.

4. Especificación de los Patrones de Comportamiento

En esta sección se presenta la propuesta para la especificación de PDC mediante el uso de perfiles UML, su inclusión en APPD, y su modelado utilizando RSA. En cada nivel se definen diferentes perfiles UML añadiendo restricciones OCL para eliminar ambigüedades. Cada perfil del nivel superior puede aplicarse al perfil del nivel siguiente. Los perfiles luego pueden ser empleados tanto en diagramas de clase como diagramas de secuencia, incorporando los estereotipos que describen a un patrón. De esta manera se puede verificar tanto los rasgos estructurales como de comportamiento y establecer una relación entre ellos.

En cada nivel se definen diferentes perfiles UML añadiendo restricciones OCL para eliminar ambigüedades. Cada perfil del nivel superior puede aplicarse al perfil del nivel siguiente. Los perfiles luego pueden ser empleados tanto en diagramas de clase como diagramas de secuencia, incorporando los estereotipos que describen a un patrón. De esta manera se puede verificar tanto los rasgos estructurales como de comportamiento y establecer una relación entre ellos.

Para representar cada nivel de la arquitectura se crearon diferentes perfiles que se describen a continuación.

4.1. Perfil Nivel 0

En el Nivel 0 se define un perfil que representa un patrón tipo. Dicho perfil contiene dos estereotipos: *TipoClase y TipoClasificador* que representan las clases (concretas) y los clasificadores existentes en un patrón de diseño (estructural o de comportamiento) y las restricciones necesarias para su definición. Este perfil se utiliza para precisar las características estructurales de un patrón general y se aplica como base al perfil de cualquier patrón individual a especificar (Figura 2).

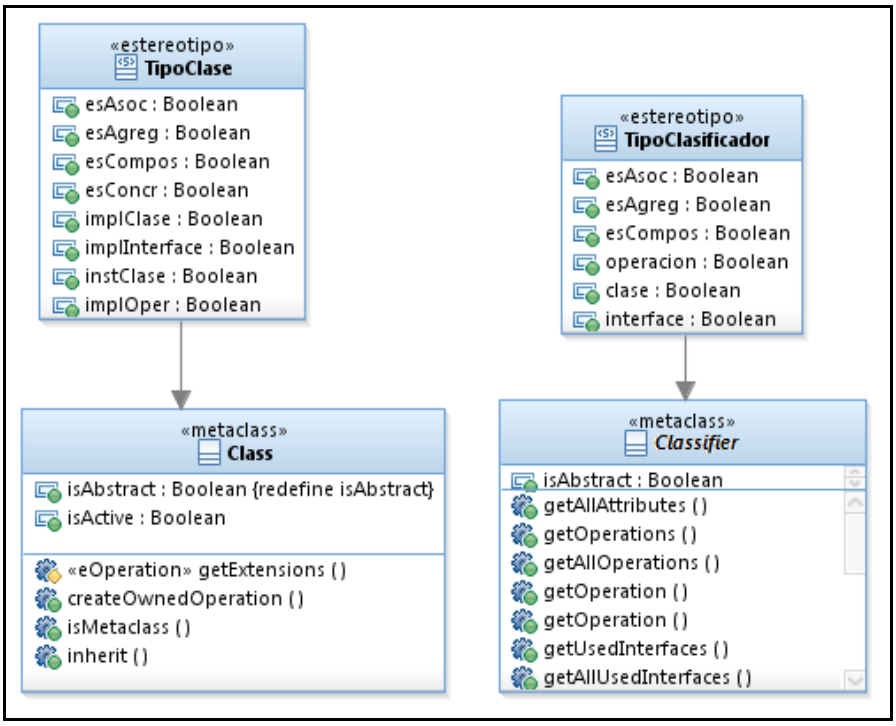

Fig. 2. Perfil DFP Nivel 0

Para definir las especificaciones dentro de un perfil se crearon atributos dentro de cada estereotipo. Estos atributos son activados si la restricción es aplicable al patrón particular tratado. En otro caso se encuentran desactivados.

Por ejemplo, los atributos *esConcr* y *esAgreg*, asociados al estereotipo *TipoClase*, son creados respectivamente, para definir clases concretas y para definir una agregación de la clase.

Una clase es concreta si el atributo *esConcr* está activado; es decir, su valor es *true*. En este caso se deben cumplir dos condiciones separadas por el operador *and*. La primera condición verifica que la clase no sea abstracta y la segunda que al menos contenga una operación que no sea abstracta. Dichas restricciones son especificadas en OCL como sigue:

```
context DesignPatternFrameworkProfile::TipoClasificador
inv:
self.esConcr implies
not self.isAbstract and (self.ownedOperation-> exists (o|not o.isAbstract))
```

4.2. Perfil Nivel 1

En la Fig. 3 se detalla el perfil incluido en el Nivel 1 de APPD. El mismo es definido para establecer las interacciones a través de los mensajes transmitidos por un conjunto de instancias para realizar una tarea. El perfil contiene dos estereotipos: *Elemento* y *Comunicador*. 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.

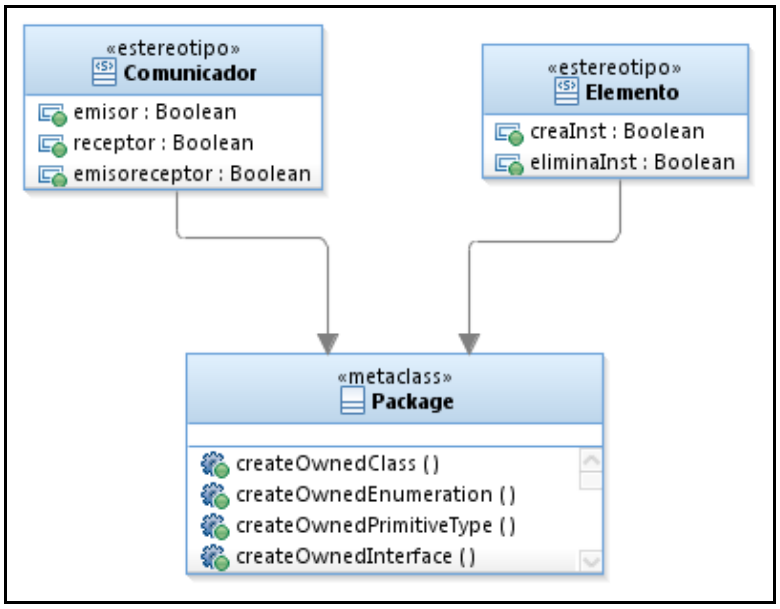

Fig. 3. Perfil de comportamiento Nivel 1

A continuación se describe parte de la especificación del perfil, detallando los atributos y restricciones OCL asociadas al estereotipo *Elemento*. Dichas restricciones, permitirán evaluar la consistencia del modelo, chequeando la relación entre el diagrama de secuencia y el diagrama de clases.

El estereotipo *Elemento* contiene dos atributos de tipo booleano que permiten definir si el patrón crea o destruye objetos: *creaInst* y *eliminaInst*. 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)
```

Entre las restricciones OCL asociadas, existe una que chequea la consistencia entre las operaciones y los mensajes; es decir, verifica que cada mensaje tenga una signatura. Para esto se seleccionan las signaturas de los mensajes y se establece que exista una correspondencia con las signaturas de las operaciones de las clases existentes, tal como se muestra a continuación.

```
context BehaviorProfile::Elemento
inv:
Message.allInstances().signature->
   forAll (s|Class.allInstances().ownedOperation
   -> exists (o1|o1.name=s.name and
      o1.ownedElement=s.ownedElement))
```

4.3. Perfil Nivel 2

En el Nivel 2 se genera un perfil para cada patrón particular de comportamiento, y se le aplican los perfiles definidos en los puntos anteriores. Siguiendo este enfoque, se conforma una librería de perfiles de patrones de comportamiento y se habilita la incorporación de nuevos patrones de diseño.
Con el objetivo de mostrar la metodología aplicada para definir cada uno de los patrones en el Nivel 2, se plantean las especificaciones correspondientes al patrón *Command* [2]. Adicionalmente, se presenta un caso de estudio en donde dicho patrón es utilizado. A través del caso de estudio se explica cómo los elementos del patrón (especificados como estereotipos) son incluidos en un diagrama de clases UML y en un diagrama de secuencia UML; y cómo es llevado a cabo el proceso de verificación y validación sobre los mismos.

Perfil UML para un Patrón de Diseño

El patrón de diseño denominado *Command* habilita el requerimiento de una operación a un objeto sin tener conocimiento sobre el contenido ni el receptor real de la misma.
Los principales elementos del patrón, según el catálogo Gof, son *Command*, *ConcreteCommand*, *Receiver e Invoker*; siendo la primera una interfaz que especifica la operación que habilita el requerimiento. Dicha operación, denominada genéricamente como *execute*, también es considerada como elemento participante del patrón. El

elemento *ConcreteCommand* implementa *execute*, mientras que e *Invoker* realiza las invocaciones al objeto *Command.*

Para introducir al patrón *Command* en APPD se define el perfil descripto en la Fig. 4. Por cada participante principal del patrón se especifica un estereotipo y, al igual que los perfiles de los niveles anteriores, se añaden restricciones OCL para establecer las características propias del patrón.

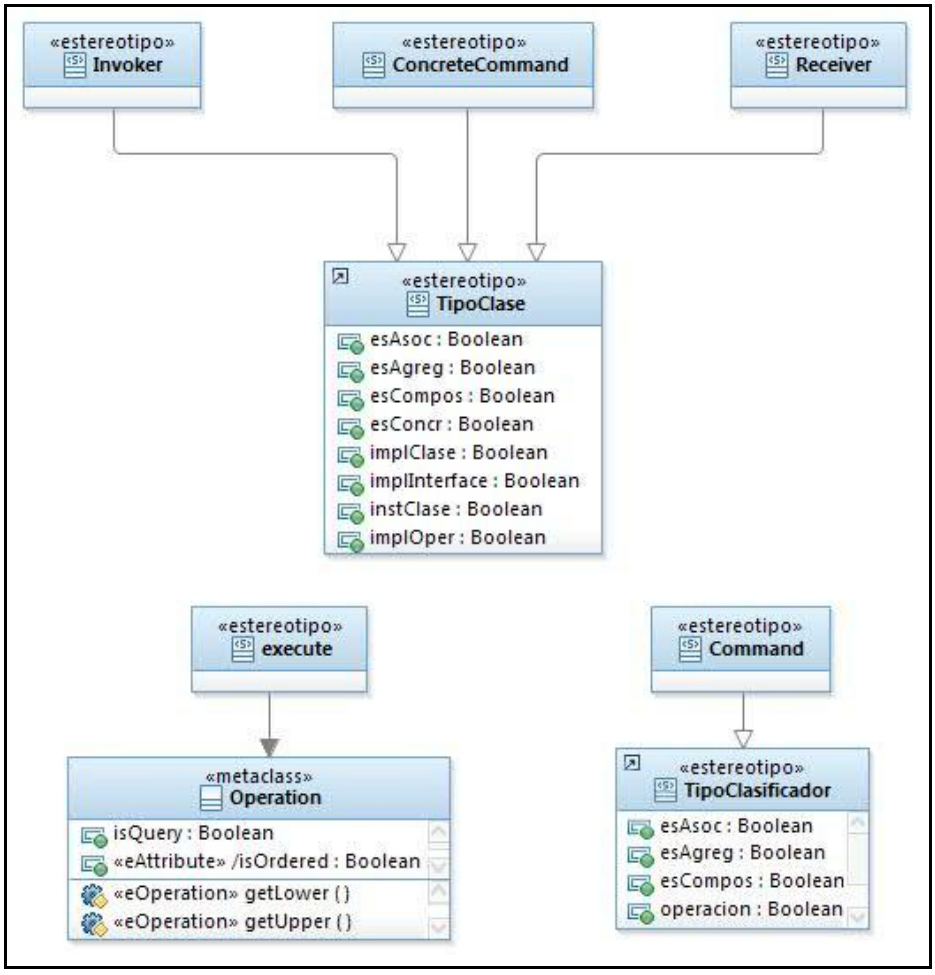

Fig. 4. Perfil del patrón Command: nivel 2

Seguidamente se detalla un caso de estudio en el que se aplica el perfil del patrón *Command* antes descripto.

Este caso es un ejemplo de comunicación entre procesos a través de sockets basado en la filosofía cliente-servidor. El servidor devuelve la nota de los estudiantes, a través la conexión socket. La información sobre los estudiantes se almacena de forma remota. Por el lado del servidor, existe una base de datos, con una aplicación muy

sencilla que contiene una tabla, donde se asocia números a instancias de la clase estudiante. El servidor recibe mensajes con números y envía mensajes que contengan datos de los estudiantes. Al implementar este caso pueden surgir algunos *problemas recurrentes* provocados por un mal diseño :

- Repetición de código: se duplica código para apertura de conexiones en cada operación.
- Mala división de responsabilidades: una clase tiene más de una responsabilidad.
- Creación de flujos de información por cada método que se usa.
- Sesión por operación: se abre una conexión cada vez que se precisa realizar una operación.

La primera solución a estos problemas es agrupar las operaciones que usan la misma conexión. Pero siguen existiendo dos responsabilidades en una clase: el manejo de los datos y el manejo de la conexión. El manejo de los datos se puede delegar a través del patrón de diseño DAO [19]. Pero queda por solucionar el manejo de las conexiones. La solución propuesta aquí es que una clase *Invocadora* se ocupe de las conexiones. Esto se logra factorizando a través del patrón *Command* como se muestra en la Fig. 5. Como se puede observar el Cliente crea una instancia de *implComando*. El *Invocador* la almacena e invoca la operación *execute* con un párametro que contiene el tipo de operación a ejecutar: apertura de conexión por ejemplo. El objeto *implComando* invoca la operación en el *Receptor*.

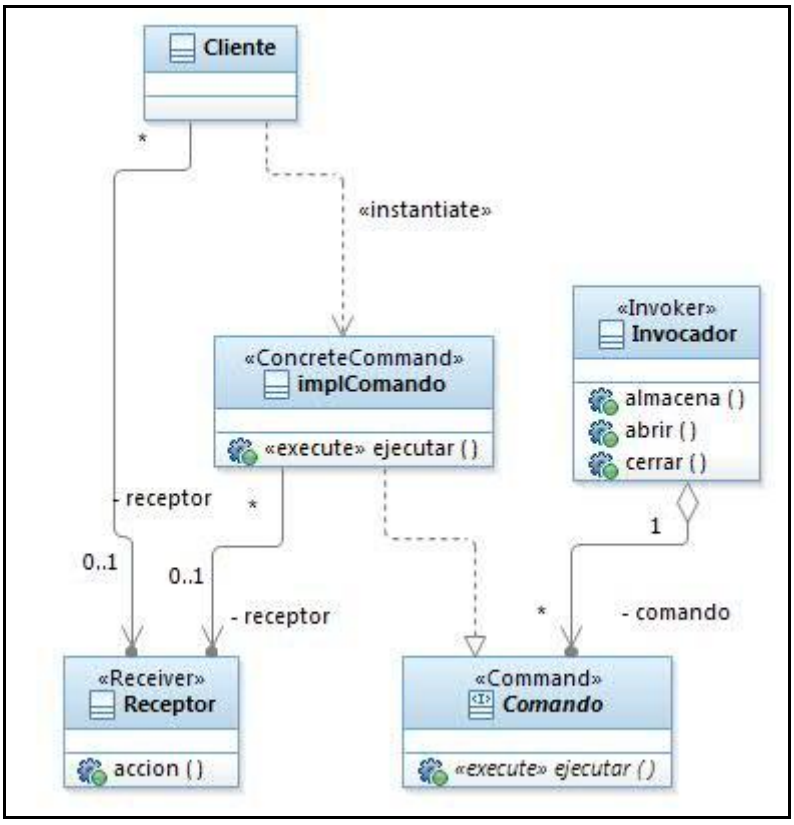

Fig. 5. Perfil *Command* aplicado a un diagrama de clases.

En el diagrama de clases se verfican las características estructurales del patrón mediante las restricciones definidas en el Nivel 2. Por ejemplo, *implComando* debe poseer una operación de tipo *execute*. En el diagrama de la Fig. 6 se muestran las interacciones entre objetos, los aspectos dinámicos, donde se desacopla el *Invocador* del *Receptor*.

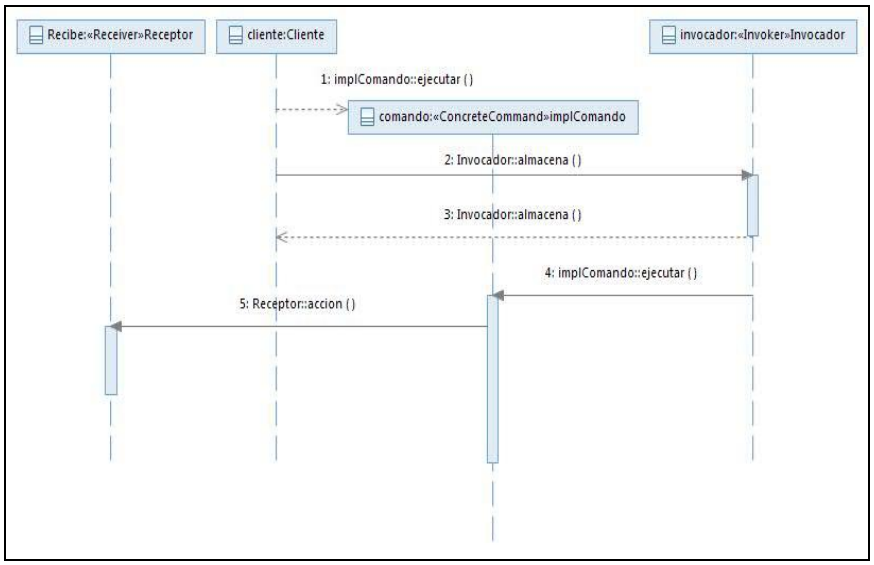

Fig. 6. Diagrama de secuencia: Creación y ejecución de Command.

En el diagrama de secuencia se verifica que existan las interacciones especificadas para el patrón. Por ejemplo un objeto de tipo *Cliente* debe enviar un mensaje de creación de un objeto tipo *ImplComand*. Además se valida la consistencia entre los diagramas de clases y de secuencia. Se comprueba entre otras cosas que por cada operación de una clase, exista un mensaje y sus signaturas se correspondan. En este caso debe existir un mensaje de solicitud tipo *execute*, que reciba de un objeto tipo *Invocador*, un mensaje de solicitud de la operación *execute*.

En la Fig. 7 se muestra otro diagrama de secuencia que también utiliza los estereotipos definidos en el perfil Command. Si se toma el diagrama de clases de la Fig. 5 al validar el modelo se generan errores como se muestra en la Fig. 8. El primer error "Se ha violado la restricción Perfil1::Elemento::ConsistenciaMensajes" se refiere a la inconsistencia entre los mensajes y las operaciones y asociaciones existentes en el diagrama de clases. El segundo error "Se ha violado la restricción Perfil1::Elemento::Creación " se produce cuando no existe una operación de creación y el atributo *CreaInst* está activado. Es decir, se comprueba la consistencia entre los diagramas. Y se verifica que el diagrama de secuencia refleja las colaboraciones especificadas para el patrón de diseño.

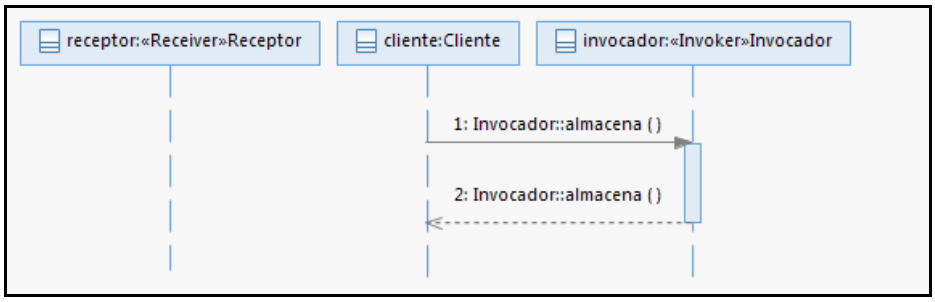

Fig. 7. Diagrama de secuencia inválido: Creación y ejecución de Command.

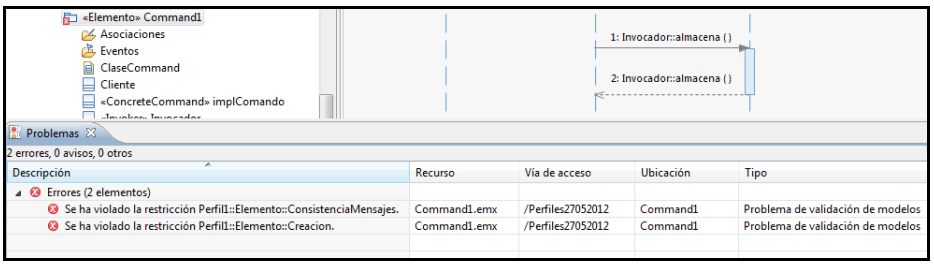

Fig. 8. Errores producidos por el diagrama de la figura 7

La misma metodología utilizada para definir *Command* ha sido aplicada para definir el resto de los patrones de comportamiento del catálogo Gof, conformando una librería de perfiles de patrones de comportamiento. Dicha librería está disponible para bajar en *https://sites.google.com/site/proyectociuda.*

5. Conclusiones

El presente trabajo ha mostrado la formalización de aspectos generales de los patrones de diseño de comportamiento a través de perfiles UML. Dicha formalización ha sido enmarcada en una arquitectura de perfiles de patrones, la cuál ha sido modelada utilizando una herramienta UML particular, Rational Software Architect. La arquitectura resultante contiene perfiles de patrones de diseño. Dichos patrones son descriptos a través de estereotipos que contienen atributos describiendo 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.

Ha sido posible encuadrar los patrones de comportamiento dentro de una arquitectura, establecer las especificaciones englobadas en los PDC y facilitar la representación de modelos. La arquitectura contiene elementos reutilizables dentro de cada per-

fil que permiten construir los perfiles PDC; es decir, los perfiles del Nivel 2 utilizan los perfiles definidos en los niveles inferiores (Nivel 1 y 0). Por lo antedicho la propuesta aquí presentada permite la mejora en el desarrollo de modelos, mediante el uso de soluciones reutilizables.

Como trabajo futuro, se intenta avanzar en el chequeo de inconsistencias de aspectos más complejos e integrar la arquitectura desarrollada con otras herramientas de modelado. Se tiene previsto también, adicionar nuevos patrones de diseño, así como enriquecer los perfiles para permitir incorporar estereotipos a otros diagramas UML, tales como diagramas de estado.

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 (1994)
3. Lagarde F., Espinoza H., Terrier F., Gérard S.: Improving UML profile design practices by leveraging conceptual domain models. 22th IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), pp. 445–448 (2007)
4. Garis, A.: Perfiles UML para la definición de Patrones de Diseño. Tesis de maestría. Universidad Nacional de San Luis (2007)
5. Queralt , A., Teniente , E.: Verification and Validation of UML Conceptual Schemas with OCL Constraints.Journal ACM Trans. Softw. Eng. Methodol., Vol. 21, No. 2 (2012)
6. 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)
7. Barotto V., Demonte M., Riesco D.: Definición de Patrones de Diseño a través del metamodelo UML.Tesis de Licenciatura en Ciencias de la Computación, Universidad Nacional de Río IV, Argentina (2005)
8. OMG: UML Superstructure, Version 2.4.1 (2011)
9. Giandini R., Pérez G., Pons, C.: A Minimal OCL-based Profile for Model Transformation. Publicado en las actas de las VI Jornadas Iberoamericanas de Ingenieria de software e Ingeniería del Conocimiento. ISBN: 978-9972-2885-2-4. Lima, Perú (2007)
10. Beydeda, S, Book, M., Volker , G.: Model-driven software development, Birkhäuser (2005)
11. Taibi, T.: Design Patterns Formalization Techniques, pp. 1-44 IGI Publishing. (2007)
12. 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)
13. Rational Software Architect. Disponible en http://www-306.ibm.com/software/rational (2008)
14. Bhutto, A.: Formal Verification of UML. Australian Journal of Basic and Applied Sciences, 5(6): 1594-1598 (2011)
15. 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)
16. 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)
17. OMG: MDA Guide versión 1.0.1 (2003)
18. OMG: Object Constraint Language, Version 2.3.1 (2012)
19. Deepak A., Crupi j., Malks D.: Core J2EE Patterns, Best Practices and Design Strategies, Reviews (2003).