Una propuesta de implementación para especificaciones de patrones de comportamiento

Alberto A. Cortez[123], Claudia A. Naveda[12]

[1]Consejo de Investigaciones -CIUDA, Universidad del Aconcagua, Mendoza, Argentina.

[2]Arquitectura de Software Dirigida por Modelos, Facultad de Ciencias Económicas San Francisco, Pontificia Universidad Católica Santa María de los Buenos Aires (UCA), Mendoza, Argentina.

[3]Instituto de Informática, Universidad de Mendoza, Mendoza, Argentina.

cortezalberto@gmail.com ,claudia_naveda@hotmail.com

Abstract. *Design Patterns were created as a tool for software reuse. Role in the Software Engineering is to provide solutions to existing problems in software development. Specifically, Patterns of Behavior (as classified by Gof) focus on describing the dynamic characteristics of systems. For an efficient implementation of these features is required clear specifications that can be validated. This paper proposes not only the modeling of these patterns using UML profiles and language restrictions OCL, but the implementation in a software tool.*

Resumen. *Los Patrones de Diseño se crearon como una herramienta para la reutilización de software. Su función dentro de la Ingeniería de Software, es brindar soluciones a los problemas existentes en el desarrollo de software. Específicamente, los Patrones de Diseño de Comportamiento (según la clasificación Gof) se enfocan en describir las características dinámicas de los sistemas. Para una implementación eficiente de dichas características se precisa especificaciones claras que se puedan validar. En este trabajo no solo se propone el modelado de dichos patrones mediante el uso de Perfiles UML y restricciones en lenguaje OCL, sino su implementación en una herramienta de software.*

1. Introducción

Para resolver un conjunto de problemas reiterados en el proceso de diseño de software se han recopilado técnicas, originadas a partir de la experiencia de especialistas. Como resultado de esta recopilación se elaboraron los denominados Patrones de Diseño. Los beneficios que provee el uso de estos patrones en el proceso de desarrollo de software incluyen: reutilización del diseño y del código potencial asociado, 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] presentaron 23 Patrones de Diseño, clasificados según dos criterios: el propósito y el ámbito. Según el propósito se catalogaron en patrones de creación (crean objetos), estructurales (componen estructuras) y de comportamiento (interacción de los objetos).

Diversos autores han investigado una notación para su formalización mediante su definición, aplicación y validación [6, 11, 16]. Los avances en esta dirección mostraron el lenguaje de modelado UML como un estándar que permite especificar y documentar sistemas [8]. Los Perfiles UML son una extensión de la sintaxis y semántica de UML. En [1, 4] se propusieron los Perfiles UML, para la especificación de Patrones de Diseño, en el marco de una *Arquitectura de Perfiles UML de Patrones de Diseño* (APPD). 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.

Este trabajo se ocupa de combinar la especificación formal de aspectos estructurales y de comportamiento para la implementación de los patrones de diseño. 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. La APPD, definida en [1, 4], es llevada a la práctica, implementándola en el contexto de una herramienta particular; en éste caso, Rational Software Architect [13]. 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 Patrones de Diseño de Comportamiento (PDC). Entre algunos de los puntos más relevantes es posible mencionar que permite utilizar herramientas UML existentes sin tener que definir nuevas, al tiempo 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. 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.

Este trabajo está organizado como sigue. La sección 2 muestra la descripción de la propuesta. La sección 3 muestra la aplicación de los conceptos mediante un ejemplo. La sección 4 expone las conclusiones y trabajos futuros.

2. Especificación de Patrones de Comportamiento

En esta sección se presenta una propuesta para la implementación de PDC mediante el uso de Perfiles UML, su inclusión en APPD, y su implementación 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. Para representar cada nivel de la arquitectura se crearon diferentes perfiles que se describen a continuación.

2.1 Perfil Nivel 0

En el Nivel 0 se define un perfil con los elementos de un patrón base. La estructura se compone de dos estereotipos: *TipoClase y TipoClasificador* que representan las clases y los clasificadores existentes en un patrón de diseño. Contienen las restricciones necesarias para su definición. (Figura 1).

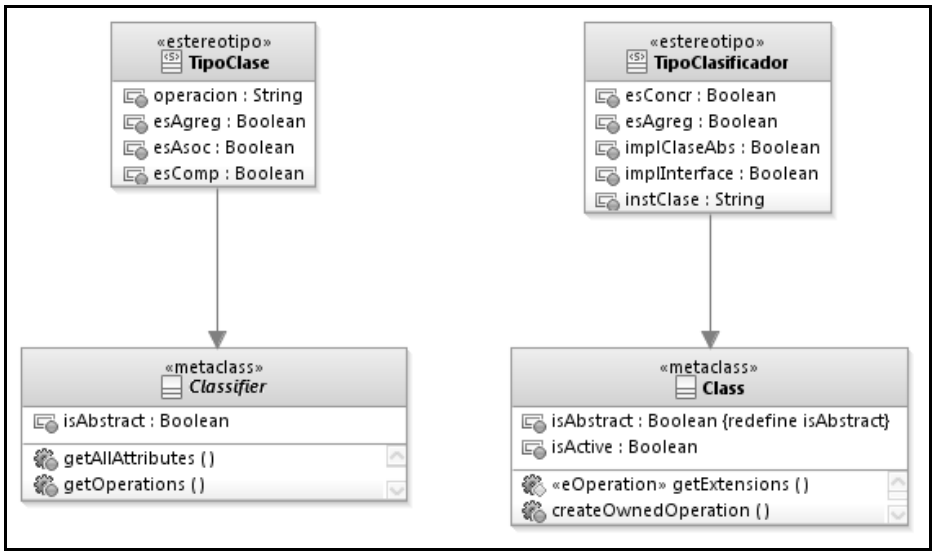

Fig. 1. Perfil DFP Nivel 0

Dentro de cada estereotipo se introdujeron atributos para activar y desactivar las restricciones. Por ejemplo, el atributo *esAgreg*, asociado al estereotipo *TipoClase*, es creado para definir una agregación de la clase.

2.2 Perfil Nivel 1

En la Fig. 2 se detalla este perfil cuyo objetivo es establecer las interacciones entre 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, que a su vez son instancias del diagrama de clases. El estereotipo *Comunicador* representa los patrones de comunicación entre objetos y permite validar las interacciones dentro de un patrón.

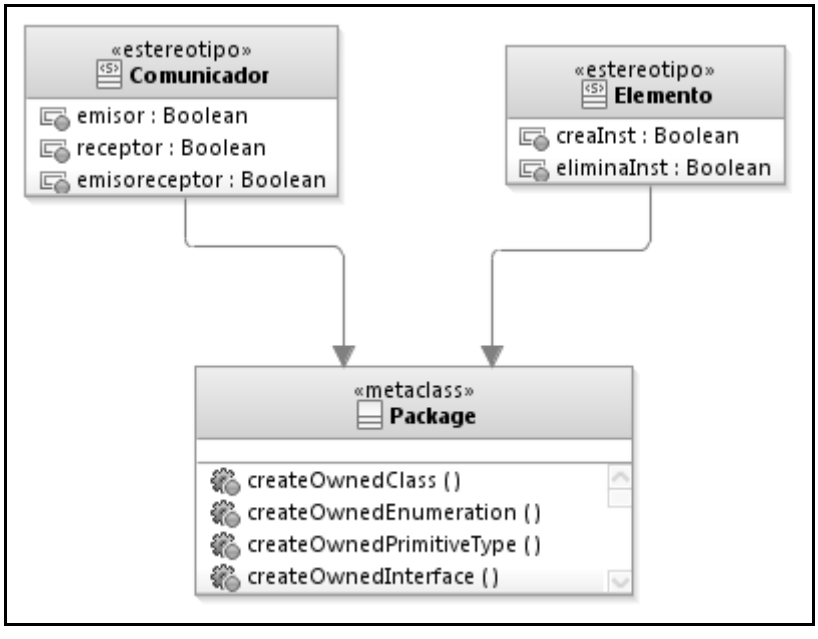

Fig. 2. Perfil de comportamiento Nivel 1

2.3 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. Más detalles sobre las especificaciones del proyecto pueden encontrarse en https://sites.google.com/site/proyectociuda.

3. Aplicación de un Perfil a un Patrón de Diseño

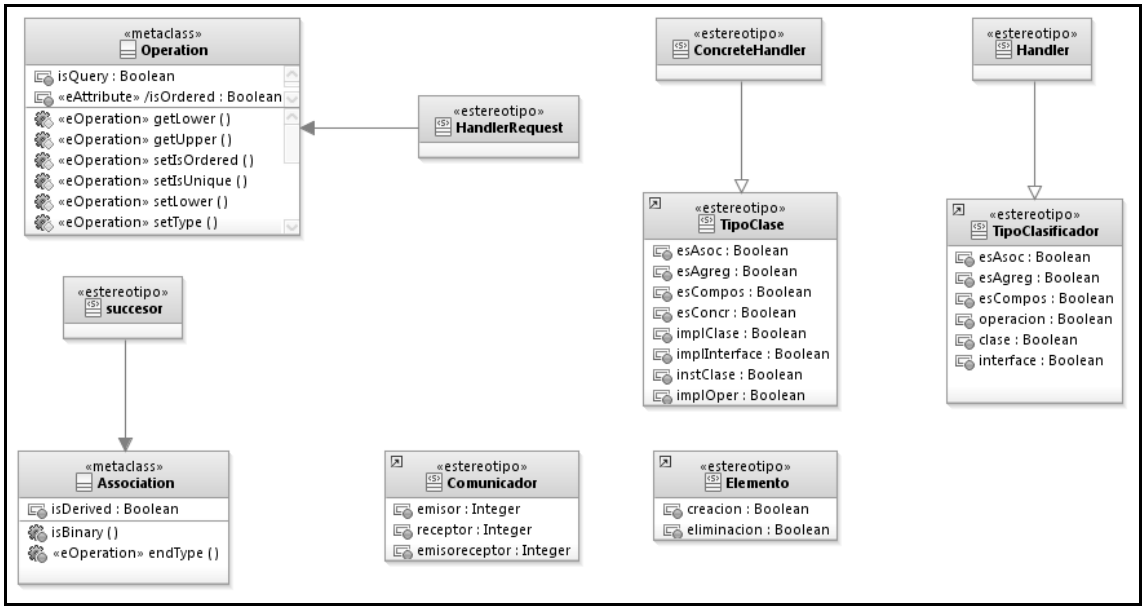

Fig. 3 Perfil Chain of Responsability

El patrón de diseño denominado *Chain of Responsability* construye una cadena de objetos receptores. Los principales elementos del patrón, según el catálogo Gof, son *Handler*, *ConcreteHandler, HandlerRequest y succesor;* siendo la primera una clase abstracta que especifica las peticiones del cliente. Dicha petición, denominada genéricamente como *HandlerRequest*, también es considerada como elemento participante del patrón. El elemento *ConcreteHandler* implementa *HandlerRequest*.Para introducir al patrón *Chain of responsability* en APPD se define el perfil descripto en la Fig. 3. 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.

Se puede visualizar mejor su aplicación a través de un caso de estudio en el que se aplica el perfil del patrón *Chain of Responsability*. 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 se muestra en la Fig. 4.

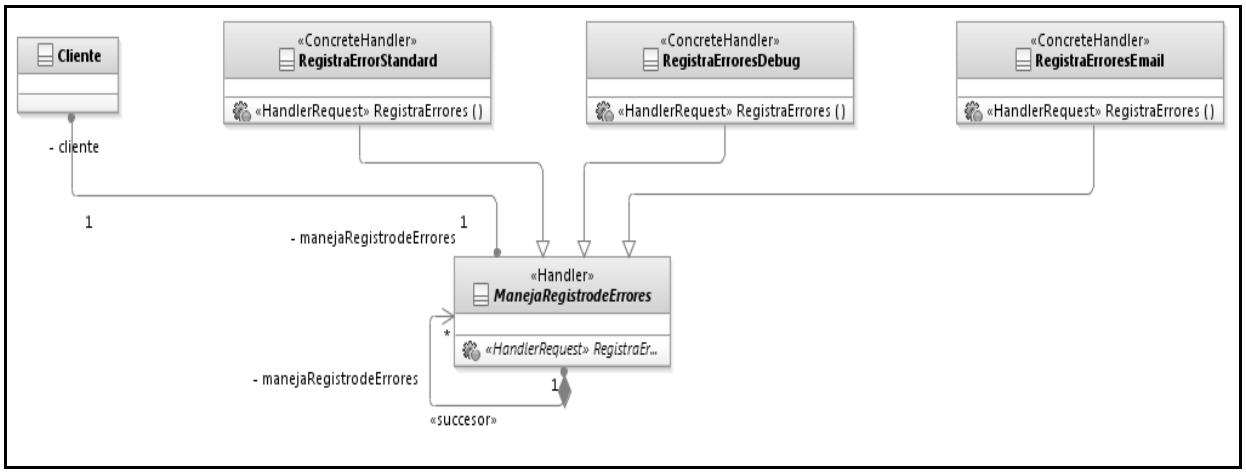

Fig. 4 Diagrama de clases: Generador de logs

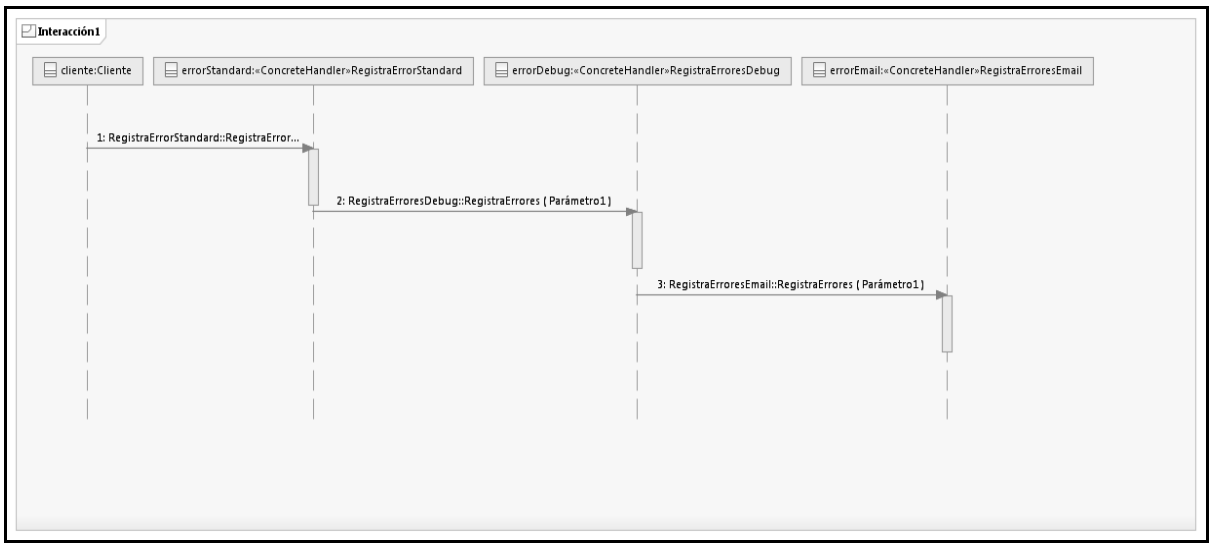

Fig. 5 Diagrama de secuencia: Generador de logs

4. Conclusiones

En este trabajo se ha mostrado la implementación de un nuevo enfoque para la formalización de Patrones de Diseño de Comportamiento. Se materializó una Arquitectura de Perfiles de Patrones, utilizando una herramienta UML particular, RSA. Con dicha estructura es posible definir perfiles de patrones de diseño, especificando tanto sus características estructurales como las de comportamiento. También es factible verificar las interacciones entre objetos, formalizar los patrones de comunicación entre ellos y validar aspectos de la consistencia entre los diagramas de clase y de secuencia. De esta manera se puede enunciar las siguientes ventajas: se ha encuadrado los PDC dentro de una arquitectura, facilitar la representación de modelos empleando los perfiles contenidos en ella e implementar las especificaciones englobadas en los PDC precisados. La propuesta aquí presentada permite la mejora en el desarrollo de modelos, mediante la detección de necesidades y la ampliación 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. Y además se espera lograr la generación de código automática.

Referencias

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)

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

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)

Garos, A.: Perfiles UML para la definición de Patrones de Diseño. Tesis de maestría. Universidad Nacional de San Luis (2007)

Queerest, A., Teniente, and E.: Verification and Validation of UML Conceptual Schemas with OCL Constraints.Journal ACM Trans. Soft. Eng. Methodol., Vol. 21, No. 2 (2012)

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)

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)

OMG: UML Superstructure, Version 2.4.1 (2011)

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)

Taibi, T.: Design Patterns Formalization Techniques, pp. 1-44 IGI Publishing. (2007)

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)

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

Bhutto, A.: Formal Verification of UML. Australian Journal of Basic and Applied Sciences, 5(6): 1594-1598 (2011)