Especificación del patrón de diseño Memento a través de un Perfil UML

Alberto Cortez, [1,2,3] Claudia Naveda [1,2,3], Ana Garis[4], Daniel Riesco[4],

[1] Departamento de Sistemas, Universidad Tecnológica Nacional , Mendoza, Argentina.

[2] Consejo de Investigaciones, Universidad del Aconcagua, Mendoza, Argentina;

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

[4] Universidad Nacional de San Luis, San Luis, Argentina.

{cortezalberto,claudialaboral}@gmail.com,{agaris, driesco}@unsl.edu.ar

Resumen. *Los patrones de diseño describen soluciones a problemas recurrentes en la ingeniería de software. En particular, los Patrones de Diseño de Comportamiento, según la clasificación Gof (en inglés, Gang of Four), abordan la especificación de las características dinámicas. Tanto los perfiles UML (en inglés, Unified Modeling Language) como el lenguaje OCL (en inglés, Object Constraint Language) se pueden utilizar para modelar y validar respectivamente, 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, son estrategias (desarrolladas a partir de la experiencia de especialistas) que permiten la documentación del software, la transmisión de conocimientos y el punto de partida para una terminología compartida. El catálogo de patrones de diseño más conocido, es el "*Gang of Four*" (GoF) [Gamma et.al 2003]. Dentro de la clasificación presentada por GoF, se agrupan los patrones por su finalidad. Se los clasifica como de Creación (crean los objetos), de Estructura (componen las estructuras) y de Comportamiento (representan la interacción entre los objetos). Los Patrones de Comportamiento describen aspectos dinámicos del sistema, como por ejemplo el flujo de control. Es esencial eliminar ambigüedades para el empleo de patrones en el proceso de modelado. Por ende es necesario elaborar especificaciones precisas para su aplicación y validación. Un enfoque para la especificación de patrones es el uso de perfiles UML. Y una de las propuestas es una Arquitectura de Perfiles UML de Patrones de Diseño (APPD) [Debnath et. al 2007].

Es una arquitectura de tres capas: Nivel 0 con las características comunes para todos los perfiles de patrones, Nivel 1 con la clasificación GoF; es decir, un perfil para las características estructurales, creacionales, de comportamiento, de clase, y de objeto; y Nivel 2, con los perfiles individuales para cada patrón. Ver Figura 1.

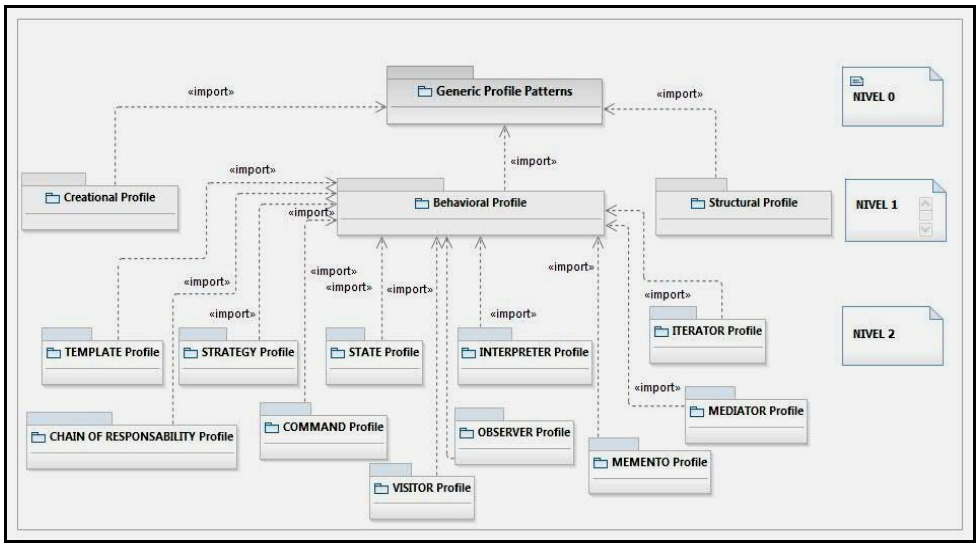


Figura 1. Arquitectura de Perfiles para Patrones de Diseño (APPD).

El presente trabajo amplía análisis anteriores de Cortez et al. [Cortez et. al 2012], para incorporar más patrones de comportamiento y aumentar la reutilización de definiciones y restricciones. Se construyeron nuevos patrones y a partir de perfiles importados, se reutilizan estereotipos de distinto nivel, con sus definiciones y restricciones.

En este artículo se muestra como el patrón memento, que tiene amplia utilidad en la industria, se puede especificar dentro de la estructura APPD. Este patrón de diseño permite capturar y exportar el estado interno de un objeto para que luego se pueda restaurar, sin romper su encapsulación. Se trata de almacenar el estado de un objeto (o del sistema completo) en un momento dado, de manera que se pueda restaurar posteriormente si fuese necesario. Para ello se mantiene almacenado el estado del objeto para un instante de tiempo en una clase independiente de aquella a la que pertenece el objeto (pero sin romper la encapsulación), de forma que ese recuerdo permita que el objeto sea modificado y pueda volver a su estado anterior. Ver Figura 2. Permite volver a estados anteriores del sistema.

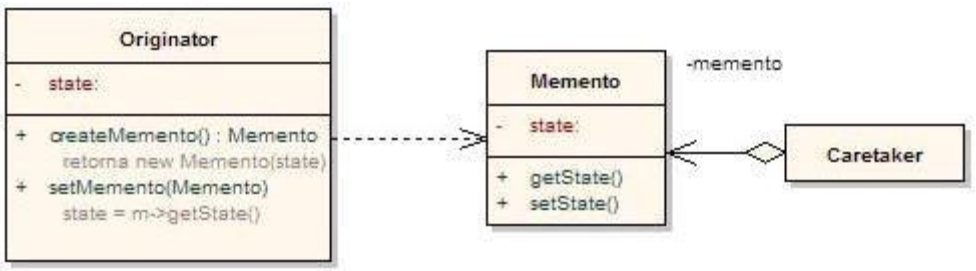


Figura 2. Arquitectura de Perfiles para Patrones de Diseño (APPD).

Caretaker: es responsable por mantener a salvo a Memento. No opera o examina su contenido.

Memento: almacena el estado interno de un objeto Originator. El Memento puede almacenar todo o parte del estado.

Originator: crea un objeto Memento conteniendo una fotografía de su estado interno.

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, a partir de la estructura APPD que contiene el Nivel 0, en el Nivel 1 y el Nivel 2 que caracteriza el patrón *Memento*. En la sección 4 se presenta un modelo donde se aplica el perfil del patrón *Memento*, validando su consistencia. En la Sección 5 se exponen las conclusiones y líneas de trabajo futuro.

2. Trabajos Relacionados

Existen dos categorías: el enfoque de modelado y el de metamodelado. El primero especifica patrones de diseño como un modelo de sistemas, un conjunto de características basados en sus comportamientos de ejecución y/o en su estructura de programación. El segundo enfoque especifica patrones de diseño como un lenguaje de metamodelado que permite especificar un conjunto de características de modelos en sí mismos.

El enfoque de modelado fue el primero en ser explorado. Existen algunas propuestas que han estudiado el uso de UML para documentar y definir patrones.

Lauder y Kent (1998) proponen una extensión a UML para lograr una " e*specifica-ción visual precisa de Patrones de Diseño*". Se aplica una notación de modelado visual para expresar modelos. Los patrones se representan con UML y "Diagramas de Restricciones". Estos últimos permiten especificar restricciones entre los elementos del modelo de objetos. Existe una notación precisa para simbolizar el comportamiento dinámico de los patrones pero su comprensión es dificultosa.

Porras y Guéhéneuc (2010) evaluaron la eficiencia de la representación de diagramas UML. Como resultado se creó un framework para comparar actuales y futuras notaciones. Su conclusión fue que era mejor utilizar estereotipos con diagramas de clases UML para identificar composición y rol de los elementos de un patrón frente al uso de anotaciones de colaboración de UML.

Lano et al. (2014) trata los patrones como transformaciones desde soluciones defectuosas a soluciones mejoradas. Se especifica y diseña transformaciones modelo, junto con la aplicación de patrones ejemplo en los principales lenguajes de transformación de modelos, como ATL, QVT, GrGen.NET y otros. Los patrones permiten mejorar la modularización y eficiencia de la transformación y reducir los requisitos de almacenamiento de datos. Se define una formalización basada en metamodelos de la transformación del modelo patrones de diseño y técnicas basadas en la medición para guiar la selección de patrones. También se proporciona una evaluación de la efectividad de patrones de transformación en una variedad de estudios de casos diferentes.

Dong y Yang (2007) aportan una solución al desarrollo de software donde se combinan múltiples patrones, los elementos participantes pueden intervenir en diferentes patrones. Se presenta un perfil UML que define nuevos estereotipos, valores etiquetados y restricciones para el trazado de patrones de diseño en diagramas UML.

Con base en este perfil, se desarrolló un servicio Web (herramienta) para visualizar explícitamente patrones de diseño en diagramas UML. Con este servicio, los usuarios pueden visualizar patrones de diseño en sus aplicaciones y composiciones porque la información relacionada con patrones se puede mostrar dinámicamente. Se realiza también un experimento comparativo con los enfoques existentes para evaluar nuestro enfoque.

Garis A. et al. (2007) establecen las ventajas de utilizar los perfiles para definir, documentar y visualizar patrones de diseño. Formulan una estructura para la definición de patrones dada por los perfiles UML. Pero no se define una semántica para todos los patrones en un sólo perfil. Se describe la semántica de cada uno en forma particular. La imposición de una jerarquía entre niveles de perfiles permite el reuso de definiciones. Se propone una arquitectura de 4 capas : "Perfil de Paquete", "Perfil del Framework de Patrones de Diseño", "Perfil de Clasificación de Patrones" y por último los "Perfiles Particulares" de cada patrón. Un patrón del nivel superior adquiere las características genéricas de los niveles inferiores pero debe especificar las características propias. El presente trabajo adopta este camino. Se continúa esta investigación, contribuyendo con la definición de especificaciones de los patrones de comportamiento.

Mikkonen (1998) aplicó las técnicas de especificación de métodos formales. Se definió el método Disco que modela y especifica las interacciones de un modelo. La base de este método es la lógica temporal de acciones. La lógica temporal tiene una notación textual, pero dentro de las herramientas de Disco se puede utilizar animaciones gráficas para las especificaciones.

Taibi (2007) describe el lenguaje BPSL (en inglés, Balanced Pattern Specification Language) que incorpora la especificación formal de aspectos estructurales y de comportamiento de los patrones. Los aspectos estructurales se formulan en lógica de primer orden, y los aspectos dinámicos en lógica temporal. En el marco de este trabajo fue desarrollada una herramienta para transformar las especificaciones al lenguaje Java a partir de instancias de patrones. También investigó cómo la composición de patrones puede ser definida en un framework.

En el segundo enfoque se encuentra el *Design Pattern Modeling Language* (por sus siglas en inglés *DPML*) de Maplesden et al.(2001, 2002). En este lenguaje, las soluciones de patrones de diseño se modelan como una colección de participantes, representando características estructurales tales como clases y métodos, adicionando restricciones asociadas y dimensiones. DPML también define un conjunto de notaciones visuales para especificar los modelos de instanciación de patrones.

El Role-Base Meta-modelan Language (por sus siglas en inglés *RBML)* propuesto por France et al. (2004), tomó un enfoque estricto de metamodelado. RBML extiende UML para metamodelado en una notación símil UML. Un patrón es visto como un meta modelo, así que cada instancia del patrón es un modelo en UML. Los participantes de un patrón y las relaciones entre ellos se representan gráficamente como roles. Las restricciones se formulan en Object Constraint Language (por sus siglas en inglés OCL), y las semánticas de métodos y atributos pueden ser definidas como plantillas OCL, instanciadas para cada patrón.

Otra herramienta en esta categoría, es el Pattern Description Language (por sus siglas en inglés, *PDL)* descripta por Albin-Amiot et al. (2001). PDL es definido por un metamodelo extendido de UML y nuevamente, los patrones de diseño son especificados

en una notación gráfica. El metamodelo extendido es un modelo abstracto y debe ser instanciado con un modelo concreto que es instancia del patrón.

Lano et al. (2014) describieron 24 especificaciones de transformaciones de modelos, patrones de diseño y 5 patrones de arquitectura de transformación de modelos. Se identifican patrones de transformación, en el que la aplicación de un patrón a una transformación la reestructura, para mejorar alguna característica de calidad como modularidad o eficiencia.

3. Especificaciones de Patrones de Diseño de Comportamiento

En esta sección se explican los niveles de la arquitectura empleada para la especificación de patrones de comportamiento, como así también los estereotipos que intervienen. Ver Figura 3. Los niveles 0 y 1 son niveles generales para todos los patrones. El nivel 2 es para los patrones particulares. En este caso solo se muestra el perfil que pertenece memento por razones de espacio.

Nivel 0:

En este nivel se describen las características estructurales de los patrones de diseño (reflejadas en diagramas estáticos). Los estereotipos incorporados en este nivel extienden las metaclases del metamodelo de clases de la superestructura de UML ["UML 2.5" 2015]. *patternPackage:* Extiende la metaclase Package y define restricciones en el contexto paquete. Se validan características de un patrón general.

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

patternClass: Extiende la metaclase Class y define un tipo de clase concreto que formaliza las operaciones de *patternClassifier.*

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

patternOperation: Extiende la metaclase Operation y restringe las operaciones de una clase.

Nivel 1:

En este nivel se añaden especificaciones para los perfiles de comportamiento, que validan la coherencia entre los diagramas estáticos (de clase) y dinámicos (de secuencia). El estereotipo *patternInteraction,* extiende la metaclase *Interaction* del metamodelo de comportamiento ["UML 2.5" 2015]. 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* que 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 construye importando los perfiles del Nivel 0 y Nivel 1. Los estereotipos del Nivel 2 heredan de los estereotipos de los niveles 0 y 1, tanto propiedades como restricciones.

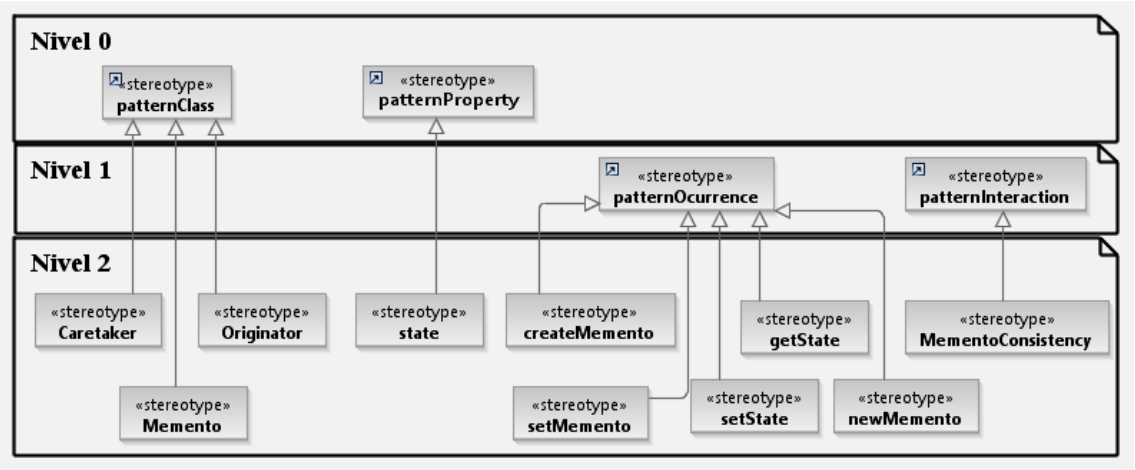


Figura 3. Características estructurales y dinámicas del patrón Memento.

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

Restricción 1: El perfil debe contener estereotipos de los participantes del patrón: Caretaker, Originator y Memento

Context PerfilMemento::MementoConsistency

let TotalEstereo : Set(String) =

Class.allInstances().getAppliedStereotypes().name->asSet(),

ConjuntoEstereo : Set(String)= Set{'Originator', 'Caretaker', 'Memento'}->asSet()

in

ConjuntoEstereo->forAll(e1| TotalEstereo->exists(e2|e1=e2))

En esta OCL, se verifica la existencia de todos los elementos del patrón dentro del modelo. Para cumplir esta restricción deben haber elementos del modelo etiquetados por dichos estereotipos.

Restricción 2: Los estereotipos setMemento y createMenento deben estar presentes en las operaciones de Originator

Context PerfilMemento::Originator

let TotalEstereo : Set(String) =

self.getAllOperations().getAppliedStereotypes().name->asSet(),

ConjuntoEstereo : Set(String)= Set{'setMemento', 'createMemento'}-> asSet()in ConjuntoEstereo->forAll(e1| TotalEstereo->exists(e2|e1=e2))

Deben existir operaciones estereotipadas con *setMemento* y *createMemento* dentro del modelo.

4. Caso de Estudio

En esta sección se aplica el patrón a un modelo. El siguiente ejemplo ilustra el uso de del comando "deshacer", que se utiliza tanto en editores de texto como en editores gráficos. El comando debe proveer un mecanismo para conservar el estado del editor, de manera de poder recuperarlo si quiero volver hacia atrás. Por lo tanto, el patrón Memento se emplea para salvar el estado de un objeto y poder recuperar los estados históricos que se necesiten (Figura 3). El memento funciona con tres clases : una que es el objeto a guardar, otra que guarda el objeto y la que agrega o recupera objetos. La clase Originador que es la clase que cambia su estado y se debe guardar; la clase Portero que se encarga de agregar los cambios del estado de Originador; y Memoria que almacena el estado de un objeto Originador. La Memoria almacena parcial o totalmente el estado de Originador. Para aplicar la especificación del patrón memento se utilizan los estereotipos definidos en el perfil: <<Originator>> ,<<Memento>> y <<Caretaker>> para el diagrama estático de clases en las clases Originador, Memoria y Portero. Además se estereotipan las operaciones con los estereotipos <<createMemento>>, <<setMemento>>, <<getState>>, <<setState>>. Como puede verse en la figura 4 .En el caso del diagrama dinámico de interacciones se estereotipa el paquete con el estereotipo <<MementoConsistency>>.

Al marcar el modelo con estos estereotipos se validan los diagramas de clases y diagramas de secuencia.

Figura 4. Diagrama de clases aplicando el patrón Memento.

El comportamiento del modelo se puede ver en el diagrama de secuencia (interacciones) de la Figura 5. La línea de vida que corresponde al objeto Main crea los objetos de tipo Portero (P1) y Originador (O1). Se llama al método setEstado() que le da un estado inicial a O1 (Originador). El objeto Main llama al procedimiento agregarMemento del objeto P1, lo que origina que el Portero realice una llamada al método createMemento() del Originador (O1). O1 (Originator) crea el Memento M1,

llamando a la operación newMemoria y establece su estado mediante el método setEstado del objeto M1 creado.

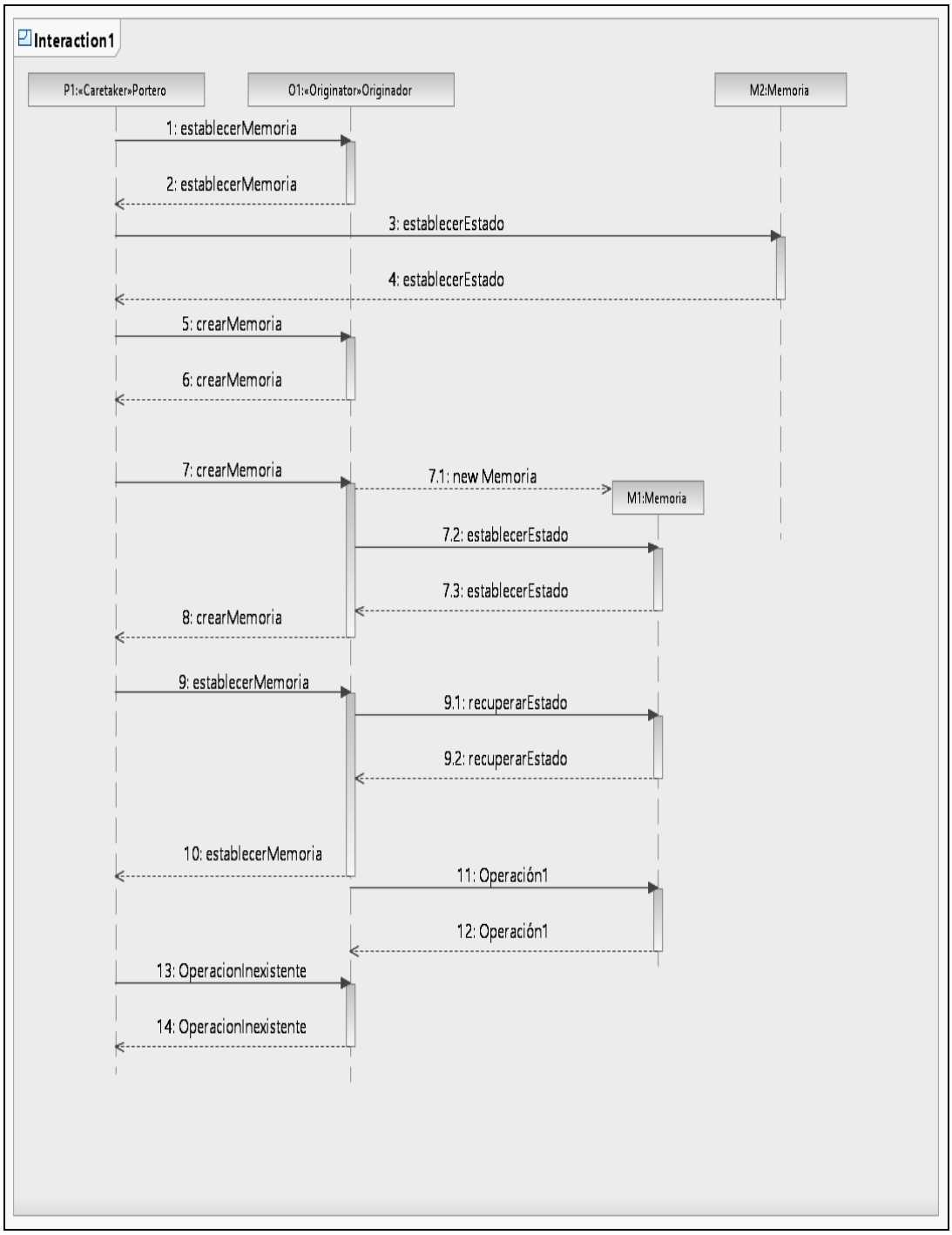
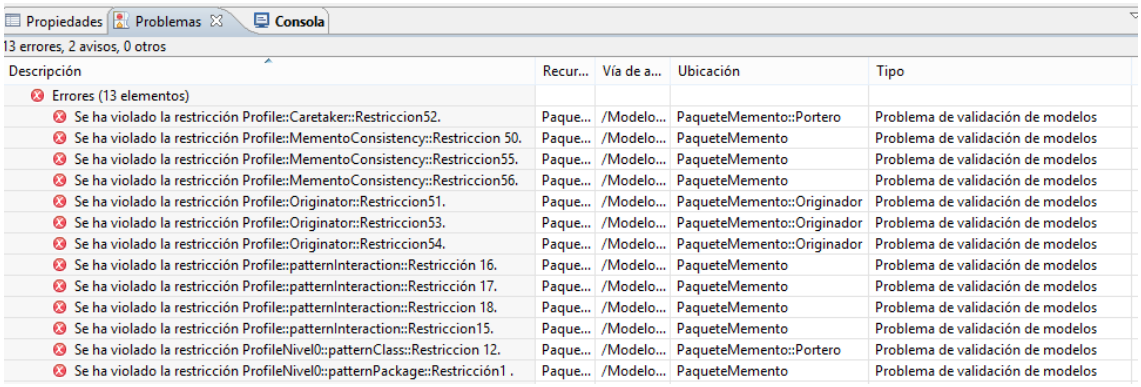


Figura 5. Diagrama de secuencia aplicando el patrón Memento.

De esta manera ya tenemos salvaguardada la memoria del estado del Objeto O1 (Originator). En el momento en que el sistema quiere recordar el estado anterior del Originador, el Main hace una llamada al método recuperarDelaMemoria() del Portero P1. Este método accede al memento M1. P1 llama al procedimiento recuperarEstado() y el procedimiento devuelve los parámetros del estado anterior. El Originador O1 queda así en su estado anterior.

. Para validar la consistencia de los diagramas se utiliza el estereotipo MementoConsistency, empleado dentro del paquete. Los diagramas de secuencia UML ejecutan las definiciones del diagrama de clases UML. Por ejemplo, la operación establecerMemoria (estereotipada con *setMemento*), en el diagrama de clases se presenta con mensajes de invocación en el diagrama de secuencia. Con la aplicación de estereotipos en diagramas UML se puede comprobar si las características del patrón se especifican correctamente. Pero también permite comprobar la consistencia entre los diagramas del modelo. Por ejemplo, una forma de comprobar consistencia entre diagramas, es comprobar que el diagrama de secuencia incluya instancias de clases y sus métodos estereotipados en el diagrama de clases.

Con la herramienta RSA ["IBM developerWorks" 2014] se detectan inconsistencias, esto es verificar la violación de las restricciones OCL propuestas en la Sección 3. Cuando termina el chequeo, la herramienta informa si los modelos cumplen las restricciones OCL que define las características del patrón. A continuación se presentan, a modo de ejemplo, algunas inconsistencias en el modelo propuesto, es decir, se validan las restricciones. Ver Figura 5.


Propiedades | Problemas | Consola

13 errores, 2 avisos, 0 otros

Descripción	Recur...	Vía de a...	Ubicación	Tipo
Errores (13 elementos)				
Se ha violado la restricción Profile::Caretaker::Restriccion52.	Paque...	/Modelo...	PaqueteMemento::Portero	Problema de validación de modelos
Se ha violado la restricción Profile::MementoConsistency::Restriccion 50.	Paque...	/Modelo...	PaqueteMemento	Problema de validación de modelos
Se ha violado la restricción Profile::MementoConsistency::Restriccion55.	Paque...	/Modelo...	PaqueteMemento	Problema de validación de modelos
Se ha violado la restricción Profile::MementoConsistency::Restriccion56.	Paque...	/Modelo...	PaqueteMemento	Problema de validación de modelos
Se ha violado la restricción Profile::Originator::Restriccion51.	Paque...	/Modelo...	PaqueteMemento::Originador	Problema de validación de modelos
Se ha violado la restricción Profile::Originator::Restriccion53.	Paque...	/Modelo...	PaqueteMemento::Originador	Problema de validación de modelos
Se ha violado la restricción Profile::Originator::Restriccion54.	Paque...	/Modelo...	PaqueteMemento::Originador	Problema de validación de modelos
Se ha violado la restricción Profile::patternInteraction::Restricción 16.	Paque...	/Modelo...	PaqueteMemento	Problema de validación de modelos
Se ha violado la restricción Profile::patternInteraction::Restricción 17.	Paque...	/Modelo...	PaqueteMemento	Problema de validación de modelos
Se ha violado la restricción Profile::patternInteraction::Restriccion 18.	Paque...	/Modelo...	PaqueteMemento	Problema de validación de modelos
Se ha violado la restricción Profile::patternInteraction::Restriccion15.	Paque...	/Modelo...	PaqueteMemento	Problema de validación de modelos
Se ha violado la restricción ProfileNivel0::patternClass::Restriccion 12.	Paque...	/Modelo...	PaqueteMemento::Portero	Problema de validación de modelos
Se ha violado la restricción ProfileNivel0::patternPackage::Restricción1 .	Paque...	/Modelo...	PaqueteMemento	Problema de validación de modelos

Figura 6. Mensajes de error en Validación de modelo Memento.

Verificaciones en el nivel 0:

Restricción 1: Se escribe una propiedad en la clase Portero sin tipo de dato.

Restricción 12: Se escribe una operación abstracta en una clase concreta Portero.

Verificaciones en el nivel 1:

Restricción 16 : Al no estereotipar la operación establecerMemoria como setMemento no coinciden las operaciones estereotipadas con las ejecuciones estereotipadas.

Restricción 17: Se crea un método Operacion1 en una clase con signatura privada y se crea un mensaje en el diagrama de secuencia para este método.

Restricción 18: Se crea un mensaje cuyo con nombre no se corresponde con ninguna operación de la clase.

Verificaciones en el nivel 2 del patrón Memento:

Restricción 50: Se confeccionó un modelo sin el participante Memento

Restricción 51: Al no estar estereotipado el participante Memento no existe su instanciación.

Restricción 53: Al no estar estereotipado el participante Memento la operación createMemento que está en Originator no tiene un parámetro estereotipado de tipo Memento.

Restricción 52: Se confeccionó un modelo sin la agregación entre el Caretaker y el Memento.

Restricción 54: No se estereotipa la operación establecerMemoria con el estereotipo setMemento.

Restricción 55: Se construye un diagrama de secuencia donde se hace una llamada al método establecerEstado antes que crearMemoria.

Restricción 56: Se construye un diagrama de secuencia donde se hace una llamada al método establecerMemoria antes que establecerEstado.

5. Conclusiones y trabajos futuros

Este trabajo resalta la importancia de la consistencia entre los diagramas UML. En el ejemplo se muestra como validar, un modelo que requiere cumplir un patrón de comportamiento. Se corrobora la coherencia entre los modelos UML estáticos y dinámicos. Para realizar esta comprobación se utilizan perfiles UML producidos sobre un esquema general. La arquitectura APPD, permite formalizar especificaciones con el lenguaje OCL, recurriendo a una herramienta UML particular, Rational Software Architect.

En este trabajo se muestra la ampliación de las definiciones de patrones de diseño de comportamiento del catálogo Gof. La lógica de primer orden es una técnica sencilla, para validar la especificación de los patrones diseño, con la aplicación de la misma, se brinda consistencia y precisión. No se usa una sintaxis especial para representar patrones; sino que se usa un artefacto UML: “el perfil”. De esta forma, el perfil se utiliza para definir un dominio específico del mismo modo que para un dominio general, como lo es la definición de patrones.

Las reutilización de buenas prácticas a través del uso apropiado de los patrones de diseño y la modelización de los sistemas mediante UML, proporcionan a los diseñadores de software una clara visualización de los sistemas y comunicación entre ellos. Se establece un vocabulario común (características propias del uso de patrones de diseño).

El trabajo realizado proporciona la base para ampliar y mejorar los instrumentos para la validación y verificación de software a través de investigaciones sobre las siguientes ideas.

- ✓ Se pueden definir transformaciones para mutar de una variante de un patrón hacia otra variante menos compleja que se pueden implementar a través del lenguaje QVT ["QVT 1.3" 2016].

- ✓ Otra arista posible es la formulación de las especificaciones en lógica de primer orden pero en lenguajes que permitan la verificación de UML y construirlas en un lenguaje como Prolog.

- ✓ Se pueden incorporar nuevos perfiles con la misma metodología de implementación, dentro de la arquitectura en cualquier nivel aprovechando el reuso de definiciones, generando nuevos estereotipos y extensiones. La ventaja de definir en el nivel de arquitectura los patrones particulares permite anexar cualquier tipo de patrón independiente de los catalogados como patrones de diseño.

- ✓ 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. Las transformaciones se pueden llevar a cabo en el lenguaje QVT ["QVT 1.3" 2016].

Referencias

Gamma, E., Acebal,C. y Lovelle, J. (2003) "*Patrones de diseño: elementos de software orientado a objetos reutilizable*." Addison-Wesley.

Debnath, N., Garis, A., Riesco, D. y Montejano, G. (2007). "Defining OCL constraints for the Proxy Design Pattern Profile," in *2007 IEEE/ACS International Conference on Computer Systems and Applications*, pp. 880–885.

Cortez, A., Garis, A. y Riesco, D. (2012). "Perfiles UML para la Especificación de Patrones de Comportamiento : Un Caso de Estudio," Proceedings of the XVIII Congreso Argentino de Ciencias de la Computación (CACIC 2012). ISBN: 978-987-1648-34-4.

Lauder, A. y Kent, S. (1998). "Precise visual specification of design patterns," in *ECOOP'98 — Object-Oriented Programming*, 1998, vol. 1445, pp. 230–236.

Cepeda Porras, G. y Guéhéneuc, Y. (2010). "An empirical study on the efficiency of different design pattern representations in UML class diagrams," *Empir. Softw. Eng.*, vol. 15, no. 5, pp. 493–5.

Lano , K., y Kolahdouz-Rahimi,S. (2014) "Model-Transformation Design Patterns," *IEEE Trans. Softw. Eng.*, vol. 40, no. 12, pp. 1224–1259.

Dong, J. ,Yang, S. y Zhang, K. (2007). "Visualizing Design Patterns in Their Applications and Compositions," *IEEE Trans. Softw. Eng.*, vol. 33, no. 7, pp. 433–453, Jul. 2007.

Mikkonen, T. (1998). "Formalizing design patterns," *Proc. 20th Int. Conf. Softw. Eng.*.

Taibi, T. (2007). "*Design Patterns Formalization Techniques*". IGI Publishing.

Maplesden, D., Hosking, J. y Grundy, J. (2001). “A visual language for design pattern modelling and instantiation,” *Proc. IEEE Symp. Human-Centric Comput. Lang. Environ. (Cat. No.01TH8587)*, pp. 2–3.

Maplesden, D., Hosking, J. y Grundy, J. (2002).“Design Pattern Modelling and Instantiation using DPML,” *Proc. Fortieth Int. Conf. Tools Pacific Objects internet, Mob. Embed. Appl. - CRPIT ’02*, vol. 10, pp. 3–11.

France, R., Kim, D., Ghosh, S. y Song, E. (2004). “A UML-based pattern specification technique,” *IEEE Trans. Softw. Eng.*, vol. 30, no. 3, pp. 193–206.

Albin-Amiot, H., Cointe, P. , Gueheneuc, Y. y Jussien, N. (2001). “Instantiating and detecting design patterns: putting bits and pieces together,” *Proc. 16th Annu. Int. Conf. Autom. Softw. Eng. (ASE 2001)*, 2001.

“UML 2.5,” *OMG*, (2015). [Online]. Available: http://www.omg.org/spec/UML/2.5/. [Accessed: 28-Jul-2015].

“IBM developerWorks : Download : IBM Rational Software Architect” *(*2014). [Online]. Available: https://www.ibm.com/developerworks/downloads/r/architect/. [Accessed: 28-Jul-2015].

"QVT 1.3" OMG (2016). [Online]. Available: https://www.omg.org/spec/QVT/1.3/

[Accessed: Jul-2016].