Una integración de Patrones de Diseño en Procesos de Ingeniería Forward de Modelos Estáticos UML

Liliana Martinez *Liliana Favre**
INTIA - Facultad de Ciencias Exactas
Universidad Nacional del Centro de la Provincia de Buenos Aires
Tandil - Argentina
{ lfavre, lmartine@exa.unicen.edu.ar }
**CIC (Comisión de Investigaciones Científicas de la Provincia de Buenos Aires)*

1. INTRODUCCIÓN

Los patrones de diseño describen soluciones a problemas de diseño recurrentes. Si bien no hay consenso sobre la forma de integrar patrones de diseño en el desarrollo de software (usando por ejemplo herramientas o lenguajes), si lo hay en cuanto a que la tarea debería ser automatizada o al menos asistida. La experiencia industrial (Beck, 1996) indica que los patrones de diseño reducen los tiempos de desarrollo, facilitan la comunicación, pero su aplicación manual es tediosa, propensa a errores y a pérdida de *traceability*.
La importancia del diseño de software a partir de patrones de diseño es ampliamente reconocida. Varios IDEs (*Integrated Development Environments*) y ambientes de modelado de software basados en UML (OMG, 2004) han comenzado a introducir soporte para patrones de diseño aunque las herramientas comerciales existentes proveen limitada asistencia para la generación de código a partir de los mismos. La mayoría simplemente asiste en un proceso "cortar y pegar", en el cual el diseñador selecciona un patrón y obtiene una pieza de código en el lenguaje apropiado. El programador necesita luego ajustar el código obtenido a la implementación (Peckham y Lloyd, 2003). En general, las técnicas empleadas no son independientes del lenguaje y son incapaces de generar código en más de un lenguaje.
Estas propuestas asumen que los patrones de diseño involucran clases dedicadas a su rol como el de colaborador dentro de un patrón de diseño particular. En general esto no es verdad, los patrones raramente existen en forma aislada. Con frecuencia un colaborador en un patrón juega un rol diferente en otro. La definición de patrón pone el énfasis en esto: un patrón es "una solución a un problema en un contexto particular" (Gamma y otros,1995). Es más, un patrón es implementado generando no sólo nuevas clases y métodos, sino adaptando el contexto existente, es decir, las construcciones de código preexistentes, a los roles que asumen en el nuevo patrón aplicado (Eden y otros, 1997).
Unas pocas herramientas CASE proveen asistencia al programador en la elección e integración de código generado automáticamente para los patrones de diseño en sus aplicaciones. Bulka (2002) analiza el estado de las herramientas de automatización de patrones de diseño y discute los pro y contras de varias propuestas. Establece que hay varios grados de automatización de patrones ofrecidos por las herramientas de modelado UML. Ellas van desde simples *templates* a patrones inteligentes. Las primeras simplemente insertan un grupo de clases relacionadas en un *workspace* (por ejemplo UMLStudio), mientras las últimas tienden a integrar el nuevo patrón insertado con las clases existentes, renombrando clases y métodos según sea requerido y respondiendo a cambios en otras partes del modelo UML (por ejemplo ModelMaker).
La mayoría de las herramientas CASE que soportan UML no proveen asistencia en la integración y generación de código de los patrones de diseño, dejando esto en manos del programador.
Teniendo en cuenta los aspectos mencionados se propone en esta investigación la integración de patrones de diseño en el proceso de ingeniería *forward* de modelos estáticos UML. La idea es definir estrategias para la detección y generación de código de patrones de diseño dentro de un

proceso riguroso que facilite el reuso y la evolución. Esta propuesta integra notaciones semiformales en UML con especificaciones algebraicas. Las transformaciones son soportadas por una librería de esquemas reusables y por un sistema de reglas de transformación, que permiten traducir paso a paso diagramas UML a especificaciones algebraicas y éstas a la vez, a código orientado a objetos. Se definieron componentes reusables específicos para asociaciones, colecciones OCL y patrones de diseño a fin de ser utilizadas en el proceso de generación de código. Para describirlos se utiliza el lenguaje NEREUS (Favre, 2003). A fin de probar la factibilidad de la propuesta se experimentará con los lenguajes orientados a objetos Eiffel y Java.
En la sección 2 se describen trabajos relacionados a esta investigación. La sección 3 describe las bases del método de ingeniería *forward* propuesto y la sección 4 considera las conclusiones.

2. TRABAJOS RELACIONADOS

Budinsky y otros (1996) presentan una herramienta para generar código de patrones de diseño en forma automática con asistencia del usuario. Esta propuesta tiene dos problemas. Por un lado el usuario debe entender qué cortar y dónde pegar y esta transformación no es reversible; una vez que el usuario ha incorporado código del patrón de diseño a su aplicación, cualquier cambio que involucre regenerar el código, lo forzará a reincorporarlo la aplicación. Por otra parte, los cambios en el código generado no pueden verse a través de la herramienta.
Florijn y otros (1997) describen un prototipo de una herramienta que soporta patrones de diseño durante el desarrollo o mantenimiento de programas orientados a objetos.
Albin-Amiot y Guéhéneuc (2001a) presentan un metamodelo que puede ser usado para obtener una representación de patrones que permite tanto la generación automática como la detección de los mismos. La propuesta apunta a la definición de patrones como entidades de modelado de primera clase. Su limitación es la integración del código del patrón generado con el código del usuario.
En Amiot-Guéhéneuc (2001b) se presentan dos herramientas que ayudan a los desarrolladores a implementar grandes aplicaciones y grandes *frameworks*, usando patrones de diseño: *Scriptor* (Generación Pura): los desarrolladores tienen nada o poco control sobre el código generado, una vez que el código es generado y entregado, no hay forma de localizar que patrón ha sido aplicado y donde; *PatternsBox* (Generación conservativa): herramienta para instanciar patrones. El principal interés radica en la conservación de todos los atributos del código fuente. Los desarrolladores necesitan escribir la mayoría o gran parte del código a mano.

3. INGENIERÍA FORWARD A TRAVÉS DE PATRONES DE DISEÑO

En esta investigación se propone integrar patrones de diseño en el proceso de ingeniería *forward* de diagramas estáticos UML. UML y OCL (Warmer y Kleppe, 2003) son usados para generar especificaciones de alto nivel en el lenguaje algebraico NEREUS. Estas especificaciones se vinculan a realizaciones específicas que se ajustan a una tecnología específica, las cuales a su vez son usadas para generar código.
NEREUS es una notación para especificar formalmente diagramas estáticos UML. La característica que distingue a este lenguaje es que está centrado en la especificación de relaciones (*relation-centric*). Permite expresar diferentes clases de relaciones como primitivas para desarrollar especificaciones. Es una notación intermedia abierta a muchos otros lenguajes formales como por ejemplo en CASL y Larch. En particular su semántica fue definida dando un significado formal a cada una de sus construcciones en términos del lenguaje CASL, que puede decirse que es actualmente el lenguaje unificado de los lenguajes algebraicos (Astesiano y otros, 2002).
Se definieron componentes reusables específicos para asociaciones, colecciones OCL y patrones de diseño a fin de ser utilizadas en el proceso de generación de código. Un componente está definido en tres niveles de abstracción: especialización, realización e implementación. El nivel de

especialización describe componentes en un alto nivel de abstracción, el cual es independiente de cualquier tecnología de implementación. Este nivel define una jerarquía de especificaciones incompletas como un grafo acíclico. El nivel de especialización tiene dos vistas. Una de ellas basada en NEREUS y la otra en UML/OCL. OCL ayuda al usuario en el proceso de identificación de componentes sin forzarlo a cambiar el estilo de especificación. Las especificaciones en el nivel de especialización están vinculadas con subcomponentes en el nivel de realización. Los subcomponentes del nivel de realización son árboles de especificaciones algebraicas: la raíz es la definición más abstracta y los nodos internos se corresponden a diferentes realizaciones de la raíz. Las especificaciones en este nivel se ajustan a una tecnología específica. El nivel de implementación asocia cada hoja del nivel de realización con diferentes implementaciones del patrón de diseño en código orientado a objetos.

Un componente reusable específico es *Association*. El nivel de especialización de este componente describe una taxonomía de asociaciones clasificadas de acuerdo a su tipo, su grado, su "navegabilidad" y multiplicidad. Cada hoja en este nivel se corresponde a subcomponentes en el nivel de realización. Por ejemplo para una asociación binaria, bidireccional y muchos a muchos, pueden asociarse diferentes realizaciones a través de *hashing*, secuencias o árboles. Los subcomponentes en el nivel de implementación expresan como implementar las asociaciones. Por ejemplo, una asociación binaria bidireccional con multiplicidad uno a uno será implementada como un atributo en cada clase asociada que contiene una referencia al objeto relacionado. Por el contrario, si la asociación es muchos a muchos, la mejor propuesta es implementar la asociación como una clase diferente, en la cual una instancia representa un vínculo y sus atributos.

Otros componentes reusables específicos fueron definidos para patrones de diseño. La Figura 1 muestra de manera gráfica y en forma parcial el componente *Observer*.

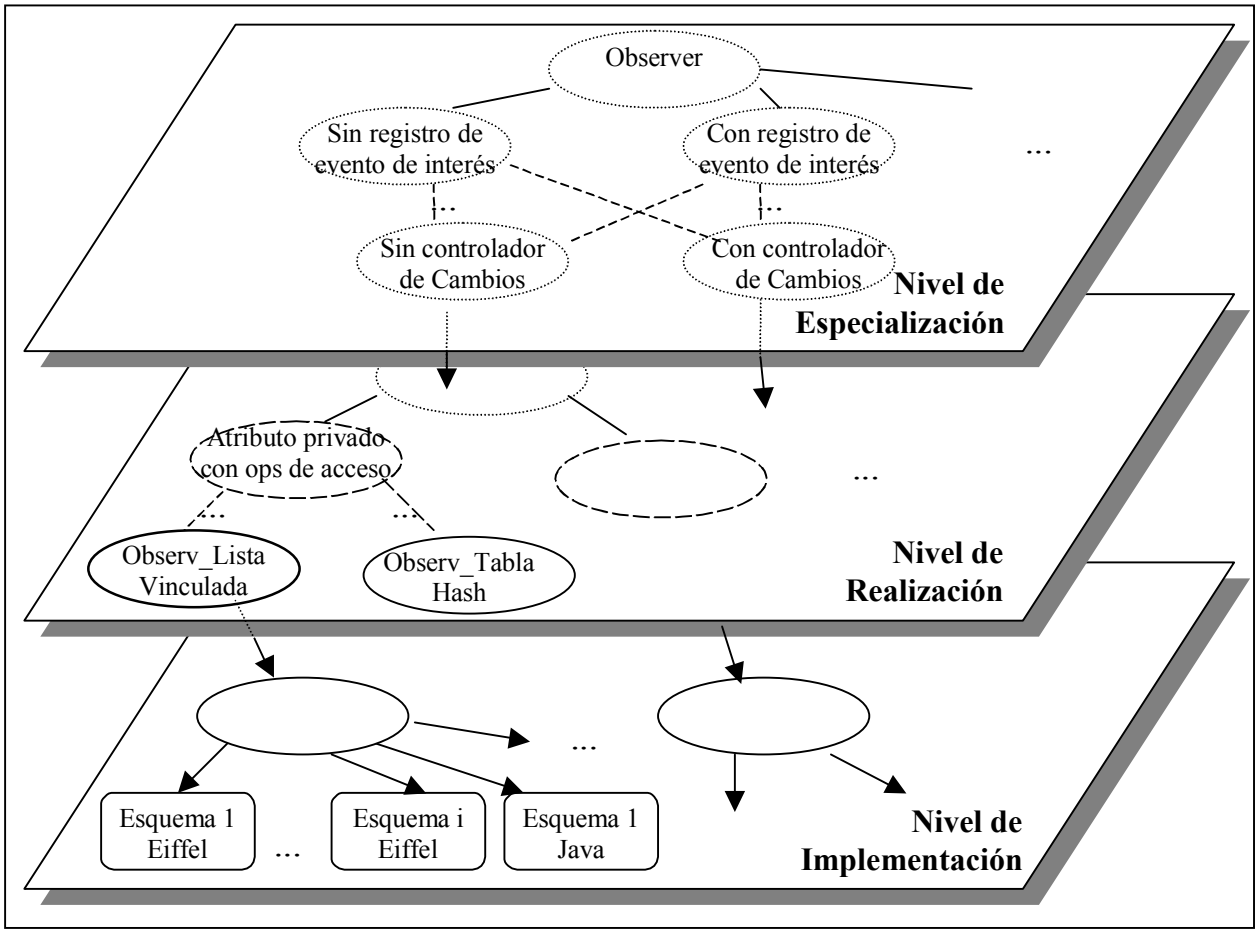

Figura 1. Componente reusable *Observer*

El reuso de componentes está basado en la aplicación de operadores de reuso: *Rename*, *Hide*, *Extend* y *Combine*. Éstos fueron definidos en los tres niveles de componentes (Favre y otros, 2003).

3.1. Generación de código a partir de patrones de diseño

La Figura 2 muestra los principales pasos del método propuesto. A partir de diagramas de clase UML, puede construirse una especificación algebraica a través de la instanciación de esquemas reusables y clases NEREUS preexistentes. Analizando las especificaciones OCL se pueden derivar axiomas en las especificaciones NEREUS. Las precondiciones escritas en OCL se usan para generar precondiciones en NEREUS. Las postcondiciones e invariantes permiten generar axiomas en NEREUS (Favre y otros 2000, Favre, 2001). De esta manera se puede construir semiautomáticamente una especificación algebraica incompleta que contiene la mayor cantidad de información que puede ser extraída del modelo UML. El refinamiento de la especificación incompleta NEREUS en la especificación algebraica completa y el código se basa en una librería de componentes reusables.
La especificación algebraica se usa para detectar patrones en el diseño por medio de un *matching* de signaturas y semántico. Se está construyendo una librería de componentes reusables para soportar este proceso. Esta contiene esquemas de patrones de diseño en NEREUS. Se tuvieron en cuenta los patrones que aparecen en Gamma y otros (1995).
Cuando un patrón de diseño es detectado, su especificación es integrada a la especificación del diagrama UML. Una implementación para el patrón de diseño es seleccionada y el código es generado e integrado con el código correspondiente al resto del diagrama UML.

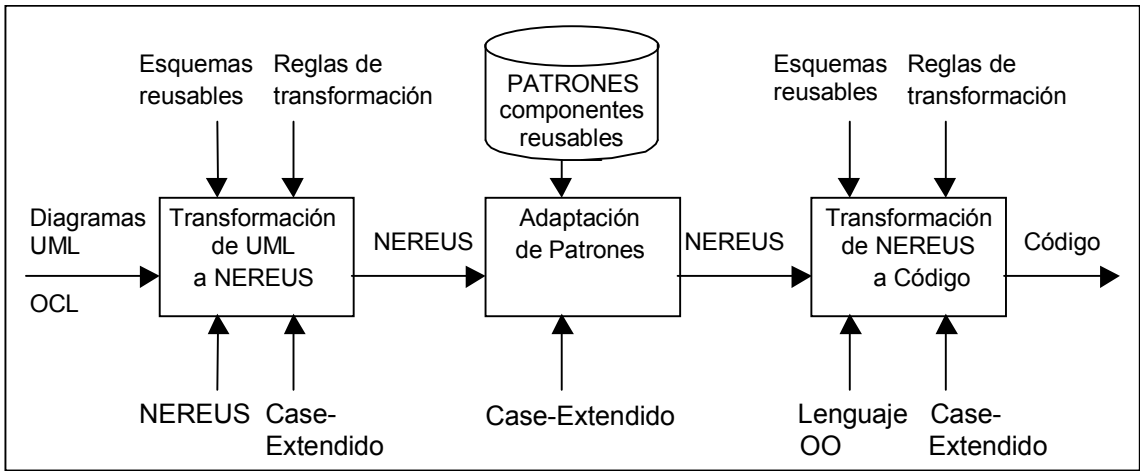

Figura 2. De UML/OCL a código

4. CONCLUSIONES

Este proyecto pretende integrar patrones de diseño en el proceso de ingeniería forward de modelos estáticos UML. Esta propuesta se basa en la integración de notaciones semiformales con técnicas algebraicas y es guiado por reglas para traducir paso a paso construcciones UML permitiendo *traceability*. Las transformaciones propuestas preservan la integridad entre especificaciones y código. La mayoría de las transformaciones se pueden deshacer, lo cual provee gran flexibilidad en el proceso de generación de código soportado por las herramientas CASE UML existentes. Siguiendo esta propuesta podemos usar las transformaciones y aplicarlas para la ingeniería reversa de código a diagramas UML.

Nuestra propuesta depende de la disponibilidad de un gran catálogo de patrones, los cuales cubren implementaciones diferentes. Actualmente, se está construyendo una librería de componentes reusables vinculados a patrones de diseño.
Un problema crucial es como detectar sub-diagramas que se correspondan con un patrón de diseño. En la actualidad, se está analizando la identificación de patrones de diseño a través de matching de signatura y matching semántico. Prevemos la integración de nuestros resultados en los ambientes de las herramientas CASE UML existentes.

REFERENCIAS

Albin-Amiot H. y Guéhéneuc Y. (2001a). Meta-modeling Design Patterns: application to pattern detection and code synthesis. *ECOOP Workshop on Automating Object-Oriented Software Development Methods*, Budapest, Hungary.

Albin-Amiot H. y Guéhéneuc Y. (2001b). Design Pattern Application: Pure-Generative Approach vs. Conservative-Generative Approach. *OOPSLA Workshop on Generative Programming*. USA.

Astesiano, E., Bidoit, M., Kirchner, H., Krieg-Brückner, B., Mosses, P., Sannella, D. y Tarlecki, A. (2002). CASL: The Common Algebraic Specification Language. *Theoretical Computer Science*, 286(2), 153-196.

Beck D., Coplien J., Crocker, R., Dominick, L., Meszaros, G. y Paulisch, F. (1996). Industrial experience with design patterns. *ICSE-18 (International Conference on Software Engineering),* Technical University of Berlin, Germany, 103- 113.

Budinsky, F., Finni, M., Vlissides, J. y Yu, P. (1996). Automatic code generation from design patterns. *IBM System Journal*, Vol 35, N° 2.

Bulka, A. (2002). Design Pattern Automation. *Third Asia-Pacific Conference on Pattern Languages of Programs (KoalaPLoP 2002)*, Melbourne, Australia.

Eden, A., Yehudai, A. y Gil, J. (1997). Precise specification and automatic application of design patterns. *1997 International Conference on Automated Software Engineering (ASE' 97),* Lake Tahoe, Canada, 143.

Favre, L. (2001) A Formal Mapping between UML Static Models and Algebraic Specifications. *Lecture Notes in Informatics (p. 7) SEW Practical UML-Based Rigorous Development Methods-Countering or Integrating the eXtremists* (Eds. Evans, R. France, A. Moreira, B. Rumpe) GI Edition, Konner Kollen-Verlag, 113-127.

Favre, L. (2003). El Lenguaje *NEREUS. Reporte interno. Grupo de Tecnología de Software, INTIA*. Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina.

Favre, L., Martinez, L. y Pereira, C. (2000). Transforming UML Static Models to Object Oriented Code. *Technology of Object-Oriented Languages and Systems, TOOLS 37, IEEE Computer Society*, 170-181.

Favre, L., Martinez, L. y Pereira, C. (2003). Forward Engineering and UML: From UML Static Models to Eiffel Code. *UML and the Unified Process* (Liliana Favre editor). Capítulo IX., IRM Press, USA, 199-217.

Florijn, G., Meijers, M. y van Winsen, P. (1997). Tool support for object-oriented patterns. *ECOOP (European Conference on Object Oriented Programming) '97*, Jyväskylä, Finland, 472-795.

Gamma, E., Helm, R., Johnson, R. y Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

OMG (2004). *Unified Modeling Language Specification*, v. 1.5. Object Management Group. Disponible en www.omg.org.

Peckham, J. y Lloyd S. (2003). Integrating Patterns into CASE Tools. *Practicing Software Engineering in the 21st Century*. Capítulo 1, IRM PRESS, 1-8.

Warmer, J. y Kleppe, A. (2003). *The Object Constraint Language Second Edition: Getting Your Models Ready for MDA*. Addison Wesley.