Controles Semánticos en el Modelamiento Orientado a Objetos

Susana Kahnert – Pablo Fillottrani
Depto. Ciencias e Ingenieríade la Computación
Universidad Nacional del Sur
Av. Alem 1253 – Bahía Blanca, Argentina
e-mail: {sak,prf }@cs.uns.edu.ar

1.- Introducción

En un sentido general, un modelo es "una abstracción de algo con el propósito de entenderlo antes de construirlo" [2]. Los modelos omiten detalles no esenciales, facilitando así el análisis de las propiedades de entidades demasiado complejas para ser entendidas directamente. Así como desde tiempos remotos las obras de ingeniería, arte o artesanía se construyen a partir de modelos, también para las distintas etapas del desarrollo de todo software resulta el Modelamiento una actividad crucial.

Es posible construir distintos modelos de un mismo sistema, cada uno enfatizando determinados aspectos que abstraen al sistema desde un punto de vista particular. Como cada uno de estos modelos enfoca la atención sobre determinados detalles mientras simplifica otros que no resulten importantes desde esa perspectiva, un sistema complejo podrá ser representado por una serie de pequeños modelos complementarios que facilitarán su entendimiento. En el proceso de desarrollo del software, un mismo modelo puede ser expresado a niveles diferentes de abstracción, según sea su propósito mostrar los requisitos y conocimiento del dominio, ayudar en el diseño del sistema, comunicar las decisiones de diseño, organizar un sistema complejo, documentar el sistema final, etc. Por consiguiente, la elección del modelo influirá profundamente en el enfoque del problema y en el aspecto de su solución.

Para que el modelo sea útil, es necesario formularlo en un lenguaje que pueda ser compartido por todas las personas involucradas en el desarrollo del sistema. En sistemas de cierta complejidad, la diversidad de personas intervinientes -Jefes de Pryecto, Analistas, Diseñadores, Programadores, Usuarios-, cada uno aportando su enfoque particular del problema, evidencia la necesidad de contar con un lenguaje formalmente definido. Una propuesta en tal sentido es UML (Lenguaje Unificado de Modelamiento) [1,3]. UML es un lenguaje gráfico para visualizar, especificar, construir y documentar los elementos de un sistema de software, es decir para definir los modelos de ese sistema. En 1995 Booch, Jacobson y Rumbaugh combinaron sus propios lenguajes intentando conservar sus ventajas particulares, y crearon la primera versión de UML. En enero de 1997 UML se propuso como un estándar ante el Object Management Group (OMG). La propuesta fue modificada, y finalmente aprobada en noviembre del mismo año, como versión 1.1. Después de varias revisiones por el OMG, surgió la versión 1.3 en 1999.

El desarrollo del Software requiere de un modelo que va evolucionando hasta llegar al sistema implementado. UML es un lenguaje para especificar ese modelo en sus distintas etapas. La notación provista por UML para expresar un modelo incluye elementos gráficos y de texto. Adosada a estos elementos, hay una interpretación semántica que intenta capturar el significado del modelo. Las herramientas de soporte para el desarrollo de sistemas orientados a objetos en UML no sólo deberán proveer de un editor gráfico para la notación, sino también ayudar a asegurar la coherencia de sus aspectos semánticos. Esta última no es una tarea sencilla porque el significado podrá cambiar de acuerdo al propósito del modelo en un momento específico. Por ejemplo, a veces puede ser deseable verificar la consistencia de las relaciones entre las clases, pero después podría ser más importante asegurar la consistencia de tipos en cada invocación a métodos

.

En el grupo de Investigación y Desarrollo de Ingeniería de Software se está actualmente trabajando en una herramienta llamada HEMOO (Herramienta para Edición de Modelos Orientados a Objetos), para soportar el desarrollo de modelos orientados a objetos, usando UML. Su objetivo es asistir al usuario en el manejo de los aspectos semánticos y notacionales del modelo. Este trabajo se enfoca especialmente en el control de los aspectos semánticos.

2. Semántica en Modelamiento orientado a Objetos

El Metamodelo UML define el lenguaje para especificar modelos, es decir, provee la descripción formal de los elementos que conformarán el modelo, tales como clases, atributos, operaciones [4]. Dado que UML es un lenguaje de especificación, proporciona un vocabulario y un juego de reglas para combinarlo. Las unidades del vocabulario se dividen en tres categorías: los *elementos* que son abstracciones de las cosas que constituyen al modelo, las *relaciones* que conectan los diferentes elementos del modelo, y los *diagramas* que reúnen un conjunto de elementos relacionados. En particular, un diagrama es una representación gráfica de un conjunto de elementos, normalmente mostrado como un grafo conexo donde los nodos son los elementos y los arcos las relaciones. UML proporciona nueve tipos diferentes de diagramas Algunos muestran los aspectos estructurales, presentando a los componentes estáticos del modelo, e indican características que se mantienen durante el ciclo de vida de un sistema Entre ellos están los Diagramas de Clases, de Objetos, de Componentes y de Distribución. Otros presentan el comportamiento del sistema, definiendo cómo interactuan los objetos a lo largo del tiempo; son los Diagramas de Casos de Uso, de Secuencia, de Actividad, de Estados y de Colaboración.

Para construir estos diagramas, los diferentes elementos no se unen de una manera arbitraria, sino a través de reglas definidas en el Metamodelo. Además, dado que el mismo elemento puede aparecer en distintos diagramas, cada aparición del elemento en un diagrama debe ser consistente con las otras. Para contemplar estas situaciones, UML proporciona el concepto de *modelo bien formado*. Se dice que un modelo es bien formado cuando está correctamente construído usando las reglas del lenguaje, y satisface el significado semántico previsto. En otros términos, un modelo bien formado debe ser sintáctica y semánticamente correcto. Para expresarlo formalmente, se definen los *controles semánticos*.

Los chequeos semánticos pueden ser caracterizados de acuerdo a diversos aspectos, tal como se ve en la Figura 1. Esta clasificación adquiere importancia teniendo en cuenta que la herramienta deberá poner a disposición del usuario la posibilidad de efectuar chequeos sobre un ámbito determinado –diagramas, relaciones o clases-, o también pudiendo decidir si desea verificar el cumplimiento de los chequeos que afectan a determinados elementos modelo. Este trabajo se enfoca en este último aspecto, es decir a la clasificación de los controles de acuerdo a su generalidad.

criterios	categorías
Generalidad	♦ Reglas • Predefinidas • Específicas del modelo ♦ Restricciones
Ambito	♦ diagramas ♦ relaciones ♦ elementos

Figura 1 Clasificación de los controles semánticos

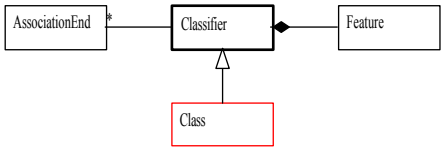
En este aspecto, las condiciones que restringen el valor de todos los elementos respectivos en el modelo se denominan *reglas,* mientras que las que afectan a items específicos se llaman *restricciones*.

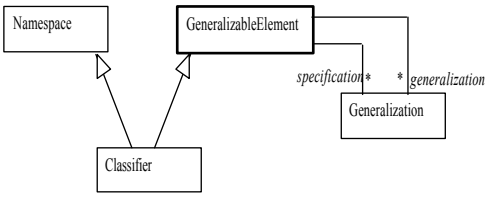
Las reglas son condiciones semánticas generales que el modelo en su conjunto debe satisfacer para ser considerado bien formado. Es posible identificar dos clases de reglas: las *predefinidas* y las *específicas del modelo*. Las reglas predefinidas son las que introduce UML para especificar cada uno de sus elementos estándares y, en general están definidas como un conjunto de invariantes de una instancia de la metaclase. Están descriptas en una combinación de lenguaje natural (semántica) y lenguaje formal (reglas de buena formación).[4]

Ejemplo:

Una de las reglas de buena formación que define UML para la metaclase `Class` establece que toda `Class` que sea concreta debe tener todas sus operaciones implementadas. Para la representación de esta regla, se requiere el atributo `isAbstract`, que pertenece a la clase `GeneralizableElement`, ancestro de `Class`. Figura 2.

En la representación en Prolog de la metaclase `Class` de UML, debe evidenciarse que `Class` desciende de `Classifier`, cuyos ancestros son `Namespace` y `GeneralizableElement`, que, a su vez descienden de `ModelElement`, que es heredero de `Element`





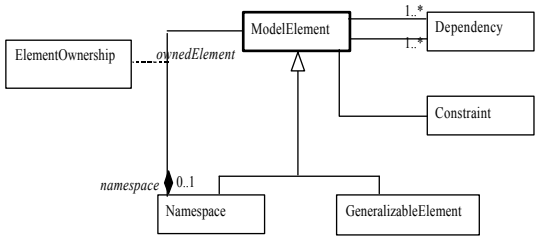
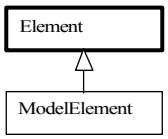


Figura 2



Las reglas específicas del modelo permiten establecer la semántica de un modelo en particular. Por ejemplo, el responsable de un determinado desarrollo podrá especificar que cada diagrama tendrá como máximo una dada cantidad de clases, o que no se admitirán clases con más de una cantidad tope de operaciones, o que todos los atributos deben ser privados.

Las restricciones son condiciones semánticas referidas a elementos específicos en el modelo. Es posible definirlas tanto en vistas estructurales como en vistas dinámicas. En el ámbito de los Diagramas de Clase, existen algunas restricciones que los objetos deben cumplir durante todo su ciclo de vida, y otras que cumplen solamente cuando interactuan con otros objetos. Las primeras se llaman *Invariantes de Clase*. Las segundas se definen sobre las operaciones que permiten la comunicación, y son las denominadas *precondiciones* y *postcondiciones*. Las precondiciones deben ser verdaderas en el momento de invocar una operación, y la responsabilidad de que esto se cumpla recae en quien la invoca. Las postcondiciones deben ser verdaderas al momento de finalizar la invocación a una operación, recayendo la responsabilidad de su satisfacción en el código que implementa esa operación.. Ambas deben ser consistentes con el conjunto de todas las demás restricciones definidas sobre la clase.

Las reglas pueden ser escogidas por el usuario para restringir en general las características de un modelo bien formado, y además para definir en particular la semántica de un modelo específico. El problema es que UML no establece, de manera clara, cómo agregar reglas específicas a un modelo existente. Con respecto a las restricciones, UML permite al usuario especificarlas en cualquier lenguaje, formal o informal. Precisamente, la importancia de herramientas como HEMOO reside en ofrecer al usuario la posibilidad de adaptar el concepto de modelo bien formado según el contexto y el momento particular en el proceso de desarrollo de software, ayudando en la identificación de posibles problemas en su definición.

El prototipo de HEMOO en el que se está trabajando está compuesto por tres subsistemas: Editor Gráfico, Editor de Propiedades, y Control Semántico. Ambos editores generan las representaciones internas para ser procesadas por el subsistema del Control Semántico.

El subsistema del Editor Gráfico tiene la responsabilidad de proveer al usuario las herramientas necesarias para definir un modelo en UML. Incluye facilidades para usar no sólo los elementos visuales, sino también los de texto, tales como restricciones, pseudocódigo y comentarios. Su implementación es en lenguaje Java. Este subsistema genera una representación interna de los elementos en un modelo que usará el subsistema del Control Semántico.

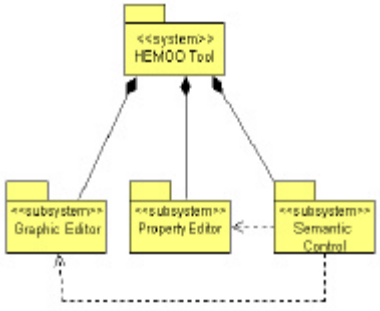


Figura 3

Para que la consistencia de las restricciones, y también su compatibilidad con las reglas semánticas sea verificada por HEMOO, la herramienta proporciona una sintaxis precisa para describirlas. Este lenguaje se llama el HEMOO Lenguaje de Restricciones (HEMOOLR), y tiene asociado una semántica formal, de manera que pueda ser controlada su consistencia..

El subsistema del Editor de Propiedades es responsable de proporcionar las herramientas necesarias para introducir en el sistema las nuevas reglas definidas por el usuario. Esto no sólo incluye el nombre, y los posibles argumentos generales (tal como una clase o una relación), sino también su definición semántica. Para introducir la definición semántica de una regla, el subsistema permite al usuario especificar todas las propiedades existentes, predefinidas y definidas por el usuario, combinadas con elementos de programación lógica tales como conjunción, disyunción, negación por falla y predicados de segundo orden. La definición de propiedades así generada se usará en el Subsistema del Control Semántico.

El subsistema del Control Semántico es responsable de inducir al usuario a especificar las propiedades que quiere sean verificadas, e informar los resultados. Usa la representación del modelo y las propiedades generada por los otros dos subsistemas.

En una primera etapa, en esta línea de investigación, la herramienta estará enfocado a la Vista Estructural del sistema, y en particular a los Diagramas de Clase, que contiene clases, interfaces, paquetes, y las relaciones entre ellos. Una vez completada esta etapa, se planea extender el alcance a todos los Diagramas involucrados en la Vista Estructural.

3. Controles Semánticos Estructurales

La implementación de los controles semánticos se hace en lenguaje Prolog. El hecho de que este lenguaje provea una semántica bien definida para sus programas permite establecer una correspondencia directa con la semántica del modelo en desarrollo. Además, la existencia de intérpretes eficentes facilita el desarrollo de las herramientas automáticas necesarias para el control de tanto las reglas predefinidas como de las específicas del modelo. Otra ventaja es que se pueden aprovechar otro tipo de herramientas existentes como los generadores de intérpretes para gramáticas formales o la programación con restricciones (CLP).

La especificación del metamodelo UML incluye constructores abstractos –no instanciables-, cuya finalidad es la de definir estructuras compartidas y dar organización al modelo, y constructores concretos que son instanciables y reflejan los constructores del modelo. Entre los primeros se pueden mencionar los constructores `GeneralizableElement`, `ModelElement`, `Classifier`, mientras que entre los últimos se encuentran los constructores `Class`, `Attribute`, `Association`. Esta especificación, para cada uno de los constructores incluye tres aspectos:

- Sintaxis abstracta, escrita mediante notación gráfica y lenguaje natural,
- Reglas de buena formación: escritas en lenguaje formal (OCL) y lenguaje natural.
- Semántica: escrita en lenguaje natural – solo definida para los constructores concretos-.

La representación en Prolog consiste en una base de datos con los hechos que representan a cada uno de los constructores del metamodelo y un conjunto de reglas con la información necesaria para contemplar cada uno de los aspectos recién mencionados, y controlar su cumplimiento mediante el subsistema de Control Semántico.

Con respecto a las Restricciones, solo será posible controlar su correcta definición.. Se entiende por correcta definición que: las expresiones booleanas sean sintácticamente válidas, que exista concordancia de tipos, que la visibilidad de los servicios involucrados sea la adecuada, que sean relaciones entre funciones, funciones y atributos o entre atributos. En el caso de que alguno de estos controles no pueda efectuarse, por no haberse completado aún la definición de la clase, deberá generar las *warnings* correspondientes.

El control de la correcta definición tanto de las Reglas específicas del modelo, como de las Restricciones, la efectúa el Editor de Propiedades, en el momento en que éstas son ingresadas.

Una vez asegurada su correcta sintaxis, el control de la consistencia de las restricciones y las reglas, se realiza traduciendo a Prolog las reglas específicas del modelo que estén bien definidas. En este punto, para controlar la inexistencia de conflictos entre reglas y/o restricciones, y dada la generalidad de lo que queremos

chequear, se presentarán tipos básicos primitivos sobre los que será posible aplicar restricciones, y se usarán herramientas (lenguajes CLP) que permiten controlar que estas restricciones son compatibles.

Dado que el modelo evoluciona, y no siempre es deseable controlar todas las reglas simultáneamente, la herramienta provee la posiblidad de que, si bien el modelo, una vez terminado, deberá cumplir todas las reglas, el usuario dispondrá de la facilidad de seleccionar cuáles reglas chequear en cada una de las etapas del desarrollo.

Actualmente se está desarrollando un prototipo de la herramienta con la funcionalidad descripta en este trabajo. La posibilidad de extensiones futuras es amplia. Aunque los Diagramas de Clase son importantes, pueden agregarse otros tipos de diagramas para la definición de un modelo. En otra línea de investigación se está trabajando actualmente en los Diagramas de Casos de Uso En particular, como próximo paso planeamos extender el alcance de la herramienta al conjunto de todos los diagramas estructurales de UML

Referencias

[1] *The Unified Modeling Language User Guide* , Grady Booch, James Rumbaugh, Ivar Jacobson. Addison Wesley, 1999.

[2] *Object-Oriented Modeling and Design*, James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen. Prentice Hall, 1991.

[3] *The Unified Modeling Languae Reference Manual*, James Rumbaugh, Ivar Jacobson, Grady Booch. Addison Wesley, 1999.

[4] *OMG Unified Modeling Languae Specification (draft).* Object Management Group, Inc. Versión 1.3a1, January 1999.

[5] *Applying Logic Programming Techniques in a Tool for Object-Oriented Modeling in UML*, Pablo Fillottrani, Elsa Estévez, Susana Kahnert. Proceedings Software Engineering & Knowledge Engineering, June 2001.