Una Introducción a los Perfiles UML

Lidia Fuentes y Antonio Vallecillo

Depto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga
Campus de Teatinos. E29071- Málaga (SPAIN)
e-mail: {lff,av}@lcc.uma.es

Resumen

El lenguaje de modelado UML es el estándar más utilizado para especificar y documentar cualquier sistema de forma precisa. Sin embargo, el hecho de que UML sea una notación de propósito muy general obliga a que muchas veces sea deseable poder contar con algún lenguaje más específico para modelar y representar los conceptos de ciertos dominios particulares. Los Perfiles UML constituyen el mecanismo que proporciona el propio UML para extender su sintaxis y su semántica para expresar los conceptos específicos de un determinado dominio de aplicación. En este artículo analizaremos los mecanismos de extensión que se utilizan para definir un Perfil UML. También discutiremos la utilidad y relevancia de estos Perfiles UML en el contexto del modelado guiado por arquitecturas (MDA).

1. Introducción

La creciente complejidad de los sistemas informáticos representa un reto importante para los ingenieros y arquitectos del software. De la preocupación inicial sobre la definición de la estructura y calidad del código final, se ha pasado a dedicar cada vez más tiempo, atención y esfuerzo al diseño y modelado del sistema. Los modelos proporcionan un mayor nivel de abstracción, permitiendo trabajar con sistemas mayores y más complejos, y facilitando el proceso de codificación e implementación del sistema de forma distribuida y en distintas plataformas.

Un *modelo* es una descripción de (parte de) un sistema, descrito en un lenguaje bien definido. Un lenguaje bien definido es un lenguaje con una sintaxis y semántica precisa, y que puede ser interpretado y manipulado por un ordenador.

Entre los lenguajes de modelado que define OMG (Object Management Group) el más conocido y usado es sin duda UML (Unified Modelling Language) [1]. UML es un lenguaje gráfico para especificar, construir y documentar los artefactos que modelan un sistema. UML fue diseñado para ser un lenguaje de modelado de propósito general, por lo que puede utilizarse para especificar la mayoría de los sistemas basados en objetos o en componentes, y para modelar aplicaciones de muy diversos dominios de aplicación (telecomunicaciones, comercio, sanidad, etc.) y plataformas de objetos distribuidos (como por ejemplo J2EE, .NET o CORBA).

El hecho de que UML sea un lenguaje de propósito general proporciona una gran flexibilidad y expresividad a la hora de modelar sistemas. Sin embargo, hay numerosas ocasiones en las que es mejor contar con algún lenguaje más específico para modelar y representar los conceptos de ciertos dominios particulares. Esto sucede, por ejemplo, cuando la sintaxis o la semántica de UML no permiten expresar los conceptos

específicos del dominio, o cuando se desea restringir y especializar los constructores propios de UML, que suelen ser demasiado genéricos y numerosos.

OMG define dos posibilidades a la hora de definir lenguajes específicos de dominio, y que se corresponden con las dos situaciones mencionadas antes: o bien se define un nuevo lenguaje (alternativa de UML), o bien se extiende el propio UML, especializando algunos de sus conceptos y restringiendo otros, pero respetando la semántica original de los elementos de UML (clases, asociaciones, atributos, operaciones, transiciones, etc.).

La primera opción es la adoptada por lenguajes como CWM [2], puesto que la semántica de algunos de sus constructores no coincide con la de UML. Para definir un nuevo lenguaje, sólo hay que describir su metamodelo utilizando el MOF, un lenguaje para describir lenguajes de modelado. Por otro lado, hay situaciones en las que es suficiente con extender el lenguaje UML utilizando una serie de mecanismos recogidos en lo que se denominan Perfiles UML (UML Profiles)

Cada una de estas dos alternativas presenta ventajas e inconvenientes. Así, definir un nuevo lenguaje ad-hoc permite mayor grado de expresividad y correspondencia con los conceptos del dominio de aplicación particular, al igual que cuando nos hacemos un traje a medida. Sin embargo, y aún cuando esos nuevos lenguajes ad-hoc se describan con MOF, el hecho de no respetar el metamodelo estándar de UML va a impedir que las herramientas UML existentes en el mercado puedan manejar sus conceptos de una forma natural. En general, resulta difícil decidir cuándo debemos crear un nuevo metamodelo, y cuándo en cambio es mejor definir una extensión de UML usando los mecanismos de extensión estándares definidos con ese propósito.

En este artículo nos vamos a centrar en analizar los mecanismos de extensión que se utilizan para definir un Perfil UML. Asimismo, también discutiremos la utilidad y relevancia de estos Perfiles UML en el contexto del modelado guiado por arquitecturas (MDA, Model Driven Architecture) [3], que propugna el desarrollo de software mediante la definición y transformación de modelos. Como veremos, el papel de los Perfiles UML dentro de MDA es muy importante.

Los Perfiles UML se definieron originalmente en la versión 1 de UML aunque era difícil saber si es estaban aplicando correctamente debido a que su definición era un tanto ambigua [4]. La nueva versión UML 2.0 mejora substancialmente su definición, especificándose de forma más clara las relaciones permitidas entre los elementos del modelo a especificar y el uso de las metaclases de un metamodelo dentro de un Perfil UML, como mostraremos en el presente trabajo.

Este artículo está estructurado en seis secciones. Tras esta introducción, la sección 2 presenta la arquitectura conceptual de cuatro niveles que define la OMG para modelar sistemas y y describir notaciones de modelado, entre los que se encuentran MOF y UML. Acto seguido la sección 3 presenta detalladamente los Perfiles UML tal y como se definen en UML 2.0, mientras que la sección 4 ofrece algunas recomendaciones para su definición y elaboración. Después, la sección 5 analiza el papel de los Perfiles UML dentro de MDA. Finalmente, la sección 6 resume el trabajo y presenta algunas conclusiones sobre el análisis realizado sobre los Perfiles UML.

2. Arquitectura de metamodelos definida por OMG

En la sección anterior hemos definido un *modelo* como una descripción de (parte de) un sistema, descrito en un lenguaje *bien definido*. El problema es ¿cómo se describe un lenguaje así?

Este problema está resuelto ya desde hace algún tiempo para los lenguajes textuales, cuya gramática se describe normalmente utilizando la notación BNF (Backus Naur Form). Ahora bien, para aquellos lenguajes cuya sintaxis es gráfica necesitamos algún mecanismo adicional. En este caso se está utilizando ahora el concepto de *meta-modelado*.

En general, podemos pensar que un modelo describe los elementos que pueden existir en un sistema, así como sus tipos. Por ejemplo, si definimos la *clase* "Persona" en un modelo, podremos utilizar *instancias* de dichas clases en nuestro sistema (por ejemplo, "Luis"). De igual forma, si el sistema que queremos modelar es un sistema UML, entonces los elementos que componen dicho sistema serán "Class", "Association", Package", etc. ¿Quién define esos elementos, y cómo se definen?

OMG define una arquitectura basada en cuatro niveles de abstracción que van a permitir distinguir entre los distintos niveles conceptuales que intervienen en el modelado de un sistema. Esos niveles se les denomina comúnmente con las iniciales M0, M1, M2 y M3.

- **El nivel M0 – Las instancias.** El nivel M0 modela el sistema real, y sus elementos son las *instancias* que componen dicho sistema. Ejemplos de dichos elementos son el cliente "Juan" que vive en "Paseo de la Castellana, Madrid" y ha comprado el ejemplar número "123" del libro "Ulises".

- **El nivel M1 – El modelo del sistema.** Los elementos del nivel M1 son los *modelos* de los sistemas concretos. En el nivel M1 es donde se definen, por ejemplo, los conceptos de "Cliente", "Compra" y "Libro", cada uno con sus correspondientes atributos ("dirección", "numero de ejemplar", "título", etc.). Existe una relación muy estrecha entre los niveles M0 y M1: los conceptos del nivel M1 definen las *clasificaciones* de los elementos del nivel M0, mientras que los elementos del nivel M0 son las *instancias* de los elementos del nivel M1.

- **El nivel M2 – El modelo del modelo (el *metamodelo*)**. Los elementos del nivel M2 son los lenguajes de modelado. El nivel M2 define los elementos que intervienen a la hora de definir un modelo del nivel M1. En el caso de un modelo UML de un sistema, los conceptos propios del nivel M2 son "Clase", "Atributo", o "Asociación". Al igual que pasaba entre los niveles M0 y M1, aquí también existe una gran relación entre los conceptos de los niveles M1 y M2: los elementos del nivel superior definen las *clases* de elementos válidos en un determinado modelo de nivel M1, mientras que los elementos del nivel M1 pueden ser considerados como *instancias* de los elementos del nivel M2.

- **El nivel M3 – El modelo de M2 (el *meta-metamodelo*)**. Finalmente, el nivel M3 define los elementos que constituyen los distintos lenguajes de modelado. De esta forma, el concepto de "clase" definido en UML (que pertenece al nivel M2) puede

verse como una instancia del correspondiente elemento del nivel M3, en donde se define de forma precisa ese concepto, así como sus características y las relaciones con otros elementos (p.e., una clase es un *clasificador*, y por tanto puede tener asociado un comportamiento, y además dispone de un conjunto de *atributos* y de *operaciones*).

OMG ha definido un lenguaje para describir los elementos del nivel M3, que se denomina MOF (Meta-Object Facility) [5]. MOF puede considerarse como un lenguaje para describir lenguajes de modelado, como pueden ser UML, CWM (Common Warehouse Metamodel), o incluso el propio MOF. Tales lenguajes pueden ser considerados como instancias del MOF, puesto que lo que proporciona el MOF es un lenguaje con los constructores y mecanismos mínimos necesarios para describir *meta-modelos* de lenguajes de modelado (es decir, los elementos que van a constituir un lenguaje dado, y las relaciones entre tales elementos).

Resumiendo, la semántica de UML viene descrita por su meta-modelo, que va a venir expresado en MOF. Si quisiéramos definir un lenguaje diferente a UML, también lo expresaríamos en MOF.

3. Perfiles UML (UML Profiles)

Para el caso en el que no queramos modificar la semántica de UML, sino sólo particularizar algunos de sus conceptos, no necesitamos definir un nuevo lenguaje utilizando el MOF. UML incluye un mecanismo de extensión en el propio lenguaje que permite definir lenguajes de modelado que son derivados de UML. De forma más precisa, el paquete Profiles de UML 2.0 define una serie de mecanismos para extender y adaptar las metaclases de un metamodelo cualquiera (y no sólo el de UML) a las necesidades concretas de una plataforma (como puede ser .NET o J2EE) o de un dominio de aplicación (tiempo real, modelado de procesos de negocio, etc.).

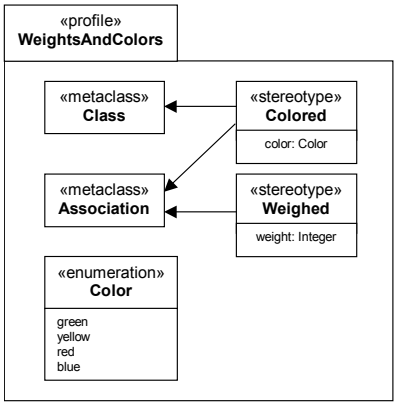
UML 2.0 señala varias razones por las que un diseñador puede querer extender y adaptar un metamodelo existente, sea el del propio UML, el de CWM, o incluso el de otro Perfil:

- Disponer de una terminología y vocabulario propio de un dominio de aplicación o de una plataforma de implementación concreta (por ejemplo, poder manejar dentro del modelo del sistema terminología propia de EJB como “Home interface”, “entity bean”, “archive”, etc.).
- Definir una sintaxis para construcciones que no cuentan con una notación propia (como sucede con las acciones).
- Definir una nueva notación para símbolos ya existentes, más acorde con el dominio de la aplicación objetivo (poder usar, por ejemplo, una figura con un ordenador en lugar del símbolo para representar un nodo que por defecto ofrece UML para representar ordenadores en una red).
- Añadir cierta semántica que no aparece determinada de forma precisa en el metamodelo (por ejemplo, la incorporación de prioridades en la recepción de señales en una máquina de estados de UML).
- Añadir cierta semántica que no existe en el metamodelo (por ejemplo relojes, tiempo continuo, etc.).

— Añadir restricciones a las existentes en el metamodelo, restringiendo su forma de utilización (por ejemplo, impidiendo que ciertas acciones se ejecuten en paralelo dentro de una transición, o forzando la existencia de ciertas asociaciones entre las clases de un modelo).
— Añadir información que puede ser útil a la hora de transformar el modelo a otros modelos, o a código.

Un Perfil se define en un paquete UML, estereotipado «profile», que extiende a un metamodelo o a otro Perfil. Tres son los mecanismos que se utilizan para definir Perfiles: estereotipos (*stereotypes*), restricciones (*constraints*), y valores etiquetados (*tagged values*).

Para ilustrar estos conceptos utilizaremos un pequeño ejemplo de Perfil UML, que va a definir dos nuevos elementos que pueden ser añadidos a cualquier modelo UML: colores y pesos.



(1) En primer lugar, un *estereotipo* viene definido por un nombre, y por una serie de elementos del metamodelo sobre los que puede asociarse. Gráficamente, los estereotipos se definen dentro de cajas, estereotipadas «stereotype». En nuestro ejemplo, el Perfil UML WeightsAndColors define dos estereotipos, Colored y Weighed, que proporcionan color y peso a un elemento UML. Tal y como se indica en el Perfil, sólo las clases y las asociaciones de UML pueden colorearse, y sólo las asociaciones pueden tener asociado un peso. Obsérvese cómo el Perfil especifica los elementos del metamodelo de UML sobre los que se pueden asociar los estereotipos estereotipados «metaclass», mediante flechas continuas de punta triangular en negrita.

(2) A los estereotipos es posible asociarles *restricciones*, que imponen condiciones sobre los elementos del metamodelo que han sido estereotipados. De esta forma pueden describirse, entre otras, las condiciones que ha de verificar un modelo "bien formado" de un sistema en un dominio de aplicación. Por ejemplo, supongamos que el metamodelo de nuestro dominio de aplicación impone la restricción de que si dos o más clases están unidas por una asociación coloreada, el color de las clases debe coincidir con el de la asociación. Dicha restricción se traduce en la siguiente restricción del Perfil UML, en el lenguaje OCL (Object Constraint Language) [6].

```
context UML::InfrastructureLibrary::Core::Constructs::Association
  inv :     self.isStereotyped("Colored") implies
        self.connection->forAll(isStereotyped("Colored") implies color=self.color)
```

Las restricciones pueden expresarse tanto en lenguaje natural como en OCL. OCL es un lenguaje para consultar y restringir los elementos de un modelo, definido por OMG y es parte integrante de UML. OCL permite escribir expresiones sobre modelos, como por ejemplo invariantes, pre- y post-condiciones, reglas de derivación para los atributos y las asociaciones, el cuerpo de las operaciones de consulta, etc. El término "Constraint" que aparece en el nombre OCL perdura de la primera versión de OCL, que sólo permitía definir restricciones. Sin embargo, la versión 2.0 de OCL proporciona un lenguaje de consultas mucho más general, con una expresividad similar a la de SQL.

(3) Finalmente, un *valor etiquetado* es un meta-atributo adicional que se asocia a una metaclase del metamodelo extendido por un Perfil. Todo valor etiquetado ha de contar con un nombre y un tipo, y se asocia un determinado estereotipo. De esta forma, el estereotipo «Weighed» puede contar con un valor etiquetado denominado "weight", de tipo Integer, y que indicará el peso de cada asociación que haya sido estereotipada como «Weighed». Los valores etiquetados se representan de forma gráfica como atributos de la clase que define el estereotipo.

Es importante señalar que estos tres mecanismos de extensión no son de primer nivel, es decir, no permiten modificar metamodelos existentes, sólo añadirles elementos y restricciones, pero respetando su sintaxis y semántica original. Sin embargo, sí que son muy adecuados para particularizar un metamodelo para uno o varios dominios o plataformas existentes. Cada una de estas particularizaciones o adaptaciones viene definida por un Perfil, que agrupa los estereotipos, restricciones, y valores etiquetados propios de tal adaptación.

Actualmente ya hay definidos varios Perfiles UML, algunos de los cuales han sido incluso estandarizados por la OMG: los Perfiles UML para CORBA y para CCM (CORBA Component Model), el Perfil UML para EDOC (Enterprise Distributed Object Computing), el Perfil UML para EAI (Enterprise Application Integration), y el Perfil UML para Planificación, Prestaciones, y Tiempo (Scheduling, Performance, and Time). Otros muchos Perfiles UML se encuentran actualmente en proceso de definición y estandarización por parte de la OMG, y se espera que vean la luz muy pronto.

También hay Perfiles UML definidos por otras organizaciones y empresas fabricantes de lenguajes de programación y herramientas que, aun no siendo estándares oficiales, están disponibles de forma pública y son comúnmente utilizados (convirtiéndose por tanto en estándares "de facto"). Un ejemplo de tales Perfiles es el "UML/EJB Mapping Specification", que ha sido definido por JCP (Java Community Process). También se dispone de Perfiles UML para determinados lenguajes de programación, como pueden ser Java o C#. Cada uno de estos Perfiles define una forma concreta de usar UML en un entorno particular. Así por ejemplo, el Perfil UML para CORBA define una forma de usar UML para modelar interfaces y artefactos de CORBA, mientras que el Perfil UML para Java define una forma concreta de modelar código Java usando UML.

4. Elaboración de Perfiles UML

En esta sección describiremos un posible método de elaboración de Perfiles UML. La especificación de UML 2.0 [7] sólo se limita a definir el concepto de Perfil y sus contenidos. Sin embargo, los autores de este trabajo también creemos que es necesario poder contar con ciertas guías y recomendaciones que sirvan de ayuda a la hora de definir un Perfil UML para una plataforma o dominio de aplicación concreto.

A la hora de definir un Perfil UML proponemos seguir los siguientes pasos:

(1) Antes de comenzar, es preciso disponer de la correspondiente definición del metamodelo de la plataforma o dominio de aplicación a modelar con un Perfil. Si no existiese, entonces definiríamos dicho metamodelo utilizando los mecanismos del propio UML (clases, relaciones de herencia, asociaciones, etc.), de la forma usual como lo haríamos si nuestro objetivo no fuese definir un perfil UML. Deberemos incluir la definición de las entidades propias del dominio, las relaciones entre ellas, así como las restricciones que limitan el uso de estas entidades y de sus relaciones.

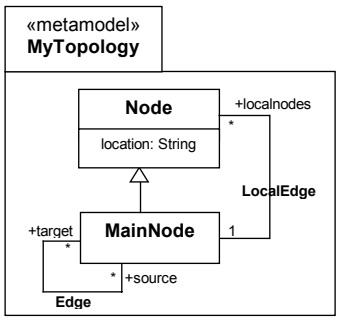
(2) Si ya disponemos del metamodelo del dominio, pasaremos a definir el Perfil. Dentro del paquete «profile» incluiremos un estereotipo por cada uno de los elementos del metamodelo que deseamos incluir en el Perfil. Estos estereotipos tendrán el mismo nombre que los elementos del metamodelo, estableciéndose de esta forma una relación entre el metamodelo y el Perfil. En principio cualquier elemento que hubiésemos necesitado para definir el metamodelo puede ser etiquetado posteriormente con un estereotipo.

(3) Es importante tener claro cuáles son los elementos del metamodelo de UML que estamos extendiendo sobre los que es posible aplicar un estereotipo. Ejemplo de tales elementos son las clases, sus asociaciones, sus atributos, las operaciones, las transiciones, los paquetes, etc. De esta forma cada estereotipo se aplicará a la metaclase de UML que se utilizó en el metamodelo del dominio para definir un concepto o una relación.

(4) Definir como valores etiquetados de los elementos del Perfil los atributos que aparezcan en el metamodelo. Incluir la definición de sus tipos, y sus posibles valores iniciales.

(5) Definir las restricciones que forman parte del Perfil, a partir de las restricciones del dominio. Por ejemplo, las multiplicidades de las asociaciones que aparecen en el metamodelo del dominio, o las propias reglas de negocio de la aplicación deben traducirse en la definición las correspondientes restricciones.

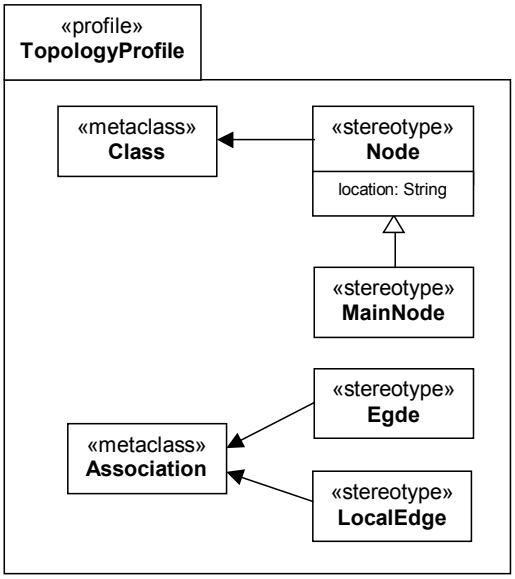
Para ilustrar este método, vamos a utilizar otro pequeño ejemplo. En primer lugar, imaginemos que queremos modelar las conexiones entre los elementos de ciertos sistemas de información según la topología en estrella, donde los nodos centrales de cada estrella pueden estar conectados entre sí. En este dominio definimos nodos (Node), conectados a través de arcos que pueden ser locales (LocalEdge) si conectan nodos de una misma estrella con su nodo central (MainNode), o bien remotos (Edge) si conectan nodos centrales entre sí. Cada nodo identifica su posición mediante una cadena de caracteres (location). Como restricción, imponemos que todos los nodos de una misma estrella compartan una misma posición geográfica. El metamodelo que describe tal dominio de aplicación puede ser descrito en UML como sigue:



Como parte de este metamodelo, también es necesario expresar el conjunto de restricciones definidas para él. En este caso, dichas restricciones pueden describirse en OCL como sigue:

```
context MyTopology::MainNode
   inv :  self.localnodes ->forAll (n : Node | n.location = self.location)
   inv:   self.target  ->forAll(n : MainNode | n.location <> self.location )
```

A partir de este metamodelo, el Perfil UML que va a representarlo va a estar descrito como un paquete UML, estereotipado «profile»:



Como puede observarse, dicho Perfil define cuatro estereotipos, correspondientes a las clases y asociaciones definidas en el metamodelo, así como las metaclases de UML sobre las que pueden aplicarse tales estereotipos. Las metaclases corresponden al metamodelo a ser extendido, en este caso al metamodelo de UML.

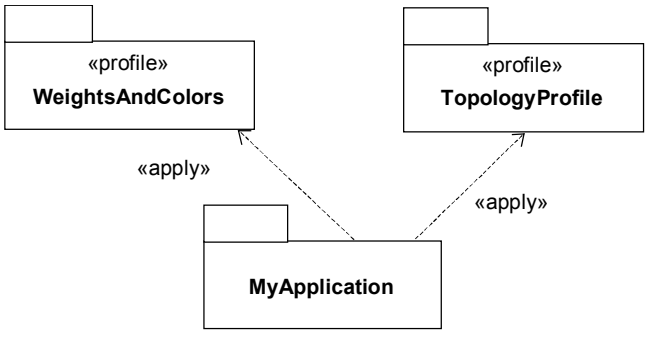
El estereotipo Node tiene asociado además un valor etiquetado (location), de tipo String.

Además de los estereotipos y los valores etiquetados, el tercer elemento de un Perfil lo constituyen las restricciones. En este caso, las restricciones del metamodelo se traducen en las siguientes restricciones del Perfil:

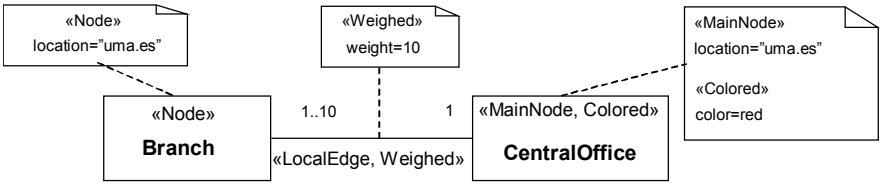
```
context UML::InfrastructureLibrary::Core::Constructs::Class
   inv :  -- Nodes should be locally connected to exactly one MainNode
          self.isStereotyped("Node") implies
                    self.connection->select(isStereotyped("LocalEdge"))->size = 1 and
                    self.connection->select(isStereotyped("Edge"))->isEmpty

context UML::InfrastructureLibrary::Core::Constructs::Association
   inv :  self.isStereotyped("LocalEdge") implies
                    self.connection->select(participant.isStereotyped("Node") or
                                             participant.isStereotyped("MainNode") ) ->
                              forAll(n1, n2 | n1.location = n2.location)

   inv :  self.isStereotyped("LocalEdge") implies
          self.connection-> exists(participant.isStereotyped("MainNode") and multiplicity.min=1
          and multiplicity.max=1)

   inv :  self.isStereotyped("Edge") implies
          self.connection->select(participant.isStereotyped("Node"))->isEmpty and
          self.connection->select(participant.isStereotyped("MainNode") ) ->
                              forAll(n1, n2 | n1.location <> n2.location)
```

Es importante observar los contextos para los que se definen dichas restricciones, que son elementos del metamodelo de UML que estamos adaptando a nuestro modelo. De esta forma, estamos añadiendo restricciones semánticas sobre ellos. La gran ventaja del uso de Perfiles es que estas restricciones van a poder comprobarse en el modelo de un sistema final conforme a un Perfil, sobre aquellas asociaciones que han sido estereotipadas con los estereotipos de dicho Perfil. De esta forma, los Perfiles UML van a describir siempre modelos "bien formados" de un sistema en un dominio de aplicación, es decir, que respeten las restricciones tanto sintácticas como semánticas que impone cada uno de los Perfiles UML que se han utilizado.

Una vez se ha definido un Perfil UML, la forma de utilizarlo en una aplicación concreta se representa mediante una relación de dependencia (estereotipada «apply») entre el paquete UML que contiene la aplicación, y los paquetes que definen los Perfiles UML que se desean utilizar. Por ejemplo, el siguiente diagrama muestra una aplicación que hace uso de los dos Perfiles UML que hemos definido con anterioridad.



Dicha aplicación puede describir entonces diagramas como el siguiente, que muestra dos clases unidas por una asociación. Ambas clases están estereotipadas como nodos de la topología en estrella. Una de ellas es además un nodo central. Obsérvese como se especifica el valor de los valores etiquetados asociados a los elementos estereotipados, mediante notas que muestran el estereotipo correspondiente, el nombre del valor etiquetado, y el valor asignado:



También se pueden mostrar valores etiquetados correspondientes a estereotipos distintos en una misma nota UML como muestra la entidad CentralOffice de la figura anterior.

5. MDA y los Perfiles UML

MDA (Model Driven Architecture) [3] es una iniciativa de la OMG que defiende la definición de modelos como elementos de primera clase para el diseño e implementación del sistema. En MDA, las actividades más importantes pasan a ser las de modelado de los diferentes aspectos de un sistema y la definición de transformaciones de un modelo a otro de forma que puedan ser automatizadas. De esta forma nos centraremos en definir modelos, no considerando detalles de implementación en una plataforma hasta el final, lo que los hace más portables, adaptables a nuevas tecnologías (p.e. .NET, J2EE, Servicios Web) e interoperables con otros sistemas independientemente de la tecnología que utilicen.

MDA define tres niveles conceptuales. En primer lugar, los requisitos del sistema se modelan en un modelo independiente de la computación (CIM, Computation Independent Model) que define el sistema en el entorno en el cual va a operar. En el siguiente nivel se encuentra el PIM (Platform Independent Model), un modelo que describe la funcionalidad del sistema, pero omitiendo detalles sobre cómo y dónde va a ser implementado (por ejemplo, el PIM puede ser independiente de la distribución y de la plataforma de objetos en donde se ejecutará, ya sea CORBA, J2EE, .NET, etc.). El objetivo es entonces transformar el PIM en un modelo dependiente de la plataforma final, denominado PSM (Platform Specific Model). De esta forma se consigue una gran independencia entre la descripción de la funcionalidad, y los detalles de implementación propios de la plataforma objetivo.

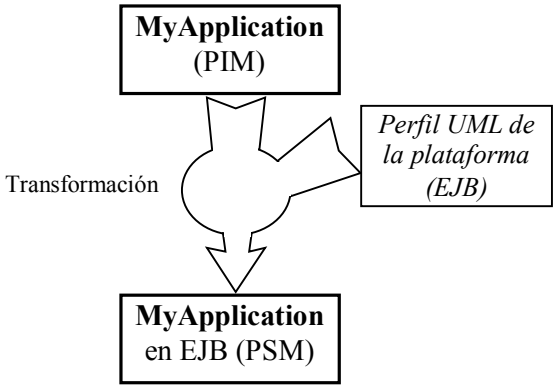
La ventaja más importante de este enfoque, es el poder definir transformaciones automáticas entre un PIM y distintos PSM, de forma que a partir del modelo PIM del sistema, de la descripción del modelo de la plataforma PSM concreta donde se implementará el sistema, y de una serie de reglas de transformación, se pueda obtener la implementación del sistema de la forma más automatizada posible.

Es precisamente a la hora de describir el modelo de la plataforma y las reglas de transformación entre modelos cuando los Perfiles UML pueden desempeñar un papel muy importante. Si usamos Perfiles UML para especificar el modelo de una plataforma, eso nos garantiza que los modelos derivados serán modelos consistentes con UML. De hecho, la principal regla para aplicar MDA con éxito es usar en lo posible estándares (modelos estándar y Perfiles UML de lenguajes o plataformas estándar). Como hemos mencionado antes, en la actualidad ya se dispone de una serie de Perfiles UML para las plataformas de componentes como CORBA, CCM, EJB, etc.

Los mecanismos que proporcionan los Perfiles UML son muy adecuados para describir los modelos de las plataformas de implementación. Se debe establecer una correspondencia (*mapping*) entre cada uno de los elementos del modelo PIM y los estereotipos, restricciones y valores etiquetados que constituyen el Perfil de la plataforma.

La idea es entonces utilizar fundamentalmente los estereotipos del Perfil de una plataforma para "marcar" los elementos de un modelo PIM de un sistema y producir el correspondiente modelo PSM, ya expresado en términos de los elementos de la plataforma destino. Una *marca* representa un concepto en el modelo PSM, y se aplica a un elemento del modelo PIM para indicar cómo debe transformarse dicho elemento según el modelo PSM destino. Las marcas podrían especificar también requisitos extra-funcionales o de calidad de servicio sobre la implementación final. Por ejemplo, algunos de los elementos del modelo PIM podrían marcarse como persistentes, o sujetos a determinadas condiciones de seguridad.

El conjunto de las marcas y las reglas de transformación que gobiernan el uso de las marcas deben ser especificadas de forma estructurada y normalmente se proporcionan junto con el Perfil UML de la plataforma destino. Si el Perfil UML de la plataforma incluye la especificación de operaciones, entonces las reglas de transformación deberían incluir también el tratamiento de dichas operaciones.



Volviendo a nuestro ejemplo, según los principios de MDA nuestro sistema MyApplication, que no incluye ningún detalle relativo a su implementación en ninguna plataforma, debería transformarse a alguna de las plataformas disponibles. En nuestro caso vamos a transformar este sistema según el Perfil UML de EJB [8] con el objeto de conseguir una implementación en la plataforma EJB/J2EE. En primer lugar tendremos que decidir para cada entidad de nuestro modelo si se transformará en una Bean o en una simple clase Java. Por ejemplo, podemos indicar que las clases estereotipadas como «Node» serán clases Java y en cambio las estereotipadas como «MainNode» se transformarán en un componente «EJBEntityBean». Dentro de este componente se incluirá la definición de las clases «EJBRemoteInterface», «EJBEntityHomeInterface» y «EJBImplementation» tal y como indica el Perfil UML del modelo EJB. Por otro lado nuestra aplicación requiere que los atributos de esta entidad se gestionen de forma persistente utilizando el modelo de contenedor de EJB. Para ello es necesario marcar los atributos que deseamos se hagan persistentes, en nuestro caso el atributo location, con el estereotipo «EJBCmpField» asignando a la etiqueta EJBPersistenceType el valor de Container que indica que la persistencia se va a gestionar por parte del contenedor.

6. Conclusiones

En este trabajo hemos presentado los Perfiles UML como un mecanismo muy adecuado para la definición de lenguajes específicos de dominio cuya semántica sea una extensión de la de UML. También hemos ilustrado la importancia de los Perfiles UML dentro del desarrollo de aplicaciones guiado por modelos (MDA). Observamos que el modelado de aplicaciones tiene cada vez mayor protagonismo frente a la codificación, y el papel de los Perfiles UML es fundamental en la definición de modelos.

La versión de UML actualmente estandarizada y soportada por herramientas comerciales es la 1.5. Sin embargo, la nueva versión 2.0 está definida y en proceso de adopción como estándar de OMG. Esta nueva versión es mucho más completa y ofrece numerosas ventajas respecto a la 1.5, mejorando muchas de sus limitaciones. Por ejemplo, presenta una mejor estructura del metamodelo, una semántica más precisa de la mayoría de sus conceptos, extensiones para manejar arquitecturas de software e intercambio de diagramas, etc.

Por otro lado, las herramientas actuales permiten definir y usar Perfiles UML únicamente de forma gráfica sin que aún soporten de forma adecuada la verificación de restricciones asociadas a estereotipos. Por lo tanto, un usuario que especifique un sistema a partir de un Perfil UML creado con una herramienta dada, no podrá estar completamente seguro de si su sistema es consistente con dicho perfil. Nos toca esperar que surjan herramientas que permitan definir Perfiles UML de acuerdo a la versión 2.0 y que permitan verificar las restricciones asociadas a los perfiles gestionando de forma adecuada la definición de nuevos estereotipos y etiquetas. Igualmente estas herramientas deberían promover la definición y uso de perfiles UML como una buena práctica para la especificación e implementación de sistemas.

Referencias

[1] ISO/IEC. Unified Modeling Language (UML). Version 1.5. International Standard ISO/IEC 19501.
[2] Object Management Group. Common Warehouse Metamodel (CWM) Specification. OMG document, ad/2001-02-01. 2001.
[3] Object Management Group. MDA Guide Version 1.0.1. OMG document, omg/2003-06-01, 2003.
[4] L. Fuentes, A. Vallecillo y J.M. Troya. Using UML Profiles for Documenting Web-Based Application Frameworks. Annals of Software Engineering, 13:249–264, 2002.
[5] Object Management Group. Meta Object Facility (MOF) Specification. OMG document: formal/2002-04-03. 2003.
[6] Object Management Group. Object Constraint Language Specification OMG document ad/02-05-09, 2002.
[7]Object Management Group. UML 2.0 Infrastructure Specification, OMG document ptc/03-09-15, 2003.
[8] Sun Java Community Process JSR-26. Public Draft. UML Profile for EJB, 2001.

Lidia Fuentes es doctora Ingeniera en Informática por la Universidad de Málaga desde 1998 y actualmente es profesora Titular de Universidad en el departamento de Lenguajes y Ciencias de la Computación en la misma Universidad. Su investigación actual se centra en el desarrollo de software basado en Componentes, marcos de trabajo, orientación a aspectos, agentes software y aplicaciones multimedia. Sus publicaciones más relevantes se encuentran en revistas internacionales de la ACM y el IEEE como IEEE Internet Computing, IEEE Transaction on Software Engineering o ACM Computing Surveys. También participa en proyectos de investigación tanto nacionales como europeos y es representante de la Universidad de Málaga en la OMG.

Antonio Vallecillo Moreno es licenciado en Matemáticas y Doctor Ingeniero en Informática por la Universidad de Málaga. Actualmente es Profesor Titular de Universidad del Departamento de Lenguajes y Ciencias de la Computación de esa Universidad, en donde también ha sido Director del Servicio Central de Informática. Aparte de su experiencia docente, su carrera profesional ha estado muy ligada a la empresa privada, en donde ha trabajado en diversas multinacionales de informática y telecomunicaciones, tanto en España como en Inglaterra. Su investigación se centra actualmente en el procesamiento abierto y distribuido, el desarrollo de software basado en componentes, y en el uso industrial de los métodos formales. Es representante de la Universidad de Málaga en AENOR y OMG, representante nacional en ISO en los temas relativos a RM-ODP, y miembro de ACM y de IEEE.