El discreto encanto del metamodelo de UML

Cómo está fundamentado el nuevo estándar para modelar y documentar sistemas

(El siguiente artículo apareció publicado en el número 60, de agosto de 1998, de la revista Soluciones Avanzadas*.)*

El tener que escoger entre distintas formas de diagramación ha hecho poco atractiva la realización de análisis y diseños de sistemas. Las discusiones en torno a cual es la "metodología" que se debe seguir no han ayudado en absoluto a popularizar la utilización de modelos en el desarrollo de sistemas. Afortunadamente, en la industria informática se ha venido forjando un amplio consenso en torno a UML como estándar de diagramación. El hecho de que el nuevo lenguaje de modelado no esté amarrado a determinadas marcas o productos nos va a facilitar la creación y el intercambio de elementos reutilizables en el análisis y el diseño.

Por Jaime González

A mediados de 1995 escuché una plática videograbada de Meilir Page-Jones donde hacía mención de "esas espantosas palabras polisilábicas que el culto de la orientación a objetos usa para sobajar y desconcertar a los que no son miembros de su cofradía". Se estaba refiriendo, por supuesto, a términos tales como el ocultamiento de información, el encapsulado, la herencia y el polimorfismo. Sin embargo, Page-Jones también nos prevenía que, independientemente de lo obscuro que nos parecieran, los conceptos abarcados por estos términos iban a impactar de ahí en adelante nuestra forma de trabajo porque la orientación a objetos no iba a ser sólo una moda pasajera.

El presente artículo no pretende sobajar a quienes no estén versados en orientación a objetos, y espero que los lectores disculpen la inclusión de uno de estos espantos polisilábicos (el *metamodelo*) en el título. En las líneas siguientes vamos a tratar de explicar qué hay detrás del amplio consenso que ha impulsado al nuevo lenguaje, que sirve para crear modelos de análisis y diseño. La importancia del tema estriba en que tenemos ante nosotros un estándar que va a afectar, tarde o temprano, la forma de realizar el desarrollo o implantación de sistemas.

El Object Management Group (OMG, conocido por sus esfuerzos de estandarización en el área de orientación a objetos) abrió un concurso en 1996 para que las empresas interesadas propusieran un lenguaje estándar de modelado de sistemas. En respuesta, el nivel 1.1 del Unified Modeling Language (UML) fue presentado y propuesto por una lista muy significativa de empresas: Hewlett-Packard, i-Logix, IBM, ICON Computing, Intellicorp, MCI Systemhouse, Microsoft, ObjecTime, Oracle, Platinum Technology, Ptech, Rational Software, Reich Technologies, Sterling Software, Softeam, Taskon y Unisys. En noviembre de ese año, como era de esperarse, la propuesta fue adoptada por el OMG. Cabe aclarar que Rational Software ya había producido versiones anteriores del UML, aunque no gozaron del grado de aceptación que tiene este último nivel.

Una de las ventajas que se pretenden lograr mediante un lenguaje unificado es permitir el intercambio de diagramas y de formas de representación de sistemas entre diversas herramientas. Por ejemplo, si un grupo de desarrolladores utilizara una herramienta CASE (*Computer Aided Software Engineering*) o una herramienta de modelado visual, debiera existir la facilidad de transportarla a otra herramienta, independientemente del proveedor o fabricante de cada una de ellas.

Pero el problema de estandarización no tiene que ver sólo con la transportabilidad de los diagramas de una herramienta de software a otra. Para mediados de los noventa las llamadas "metodologías" habían proliferado de tal forma que los proyectos y equipos de trabajo se topaban constantemente con dificultades para seleccionar un método de análisis y diseño. Los métodos propuestos por Grady Booch, Ivar Jacobson, James Martin, James Odell, Edward Yourdon, y muchos más, tenían asociadas sus

formas peculiares de diagramación. En otras palabras, cada uno de estos métodos contaba con su propio lenguaje de modelado. Esta falta de estandarización impedía la reutilización de soluciones de un proyecto a otro, y muchas veces inhibía la inversión en capacitación de personal y en herramientas para diagramar. A lo anterior debemos añadir el esfuerzo y los riesgos inherentes a la curva de aprendizaje de estos métodos de análisis y diseño.

Un paso para salir de la crisis

UML no va a ser la panacea para el desarrollo de sistemas, pero ciertamente va a contribuir a la solución de algunos de los problemas más agudos, especialmente en proyectos grandes o complejos.

En el desarrollo de software los modelos juegan el mismo papel que los planos y las especificaciones en la industria de la construcción: primero, porque representan la forma de ir plasmando las aproximaciones e ideas necesarias para resolver los requerimientos del cliente; segundo, porque los modelos son esenciales para la comunicación entre los distintos equipos de trabajo y entre las distintas disciplinas que participan en un proyecto; tercero, porque es mucho más económico hacer correcciones sobre un modelo “en papel” que hacerlas durante la construcción misma. Y existen muchas razones más, como es el caso de la facilidad que brinda la documentación para dar mantenimiento a una aplicación.

El uso de métodos y modelos tiene mucho que ver con una de las contradicciones más notorias de nuestra época: conforme se incrementa la importancia del software en las operaciones críticas de empresas e instituciones, sucede también que las mejores técnicas para producir o implantar este software son desconocidas o no se encuentran adecuadamente diseminadas. Las cifras son alarmantes. Según W. Wayt Gibbs, en su artículo "Software's Chronic Crisis" publicado en la edición de septiembre de 1994 de *Scientific American*, por cada seis grandes proyectos que son puestos en producción hay otros dos que han sido cancelados; y tres cuartas partes de los que llegan a ser puestos en marcha son considerados fracasos operativos.

Las causas de la crisis del software son variadas, pero entre las más importantes se encuentra la falta de una *cultura* común a quienes realizamos los desarrollos. Y esta carencia está íntimamente relacionada con la falta de medios de comunicación para, por ejemplo, intercambiar técnicas y soluciones exitosas. Es precisamente en estos aspectos donde UML nos puede ayudar.

Objetivos principales de UML

Uno de los objetivos principales del OMG al solicitar propuestas para el nuevo estándar era el de permitir la interoperabilidad entre distintas herramientas CASE (muchas de la cuales, dicho sea de paso, ya permiten la creación de modelos con UML). Quizás el uso más extendido de este tipo de herramientas hoy en día consiste en generar código de SQL para crear tablas y disparadores (*triggers*) a partir de los elementos representados en diagramas. La estandarización en la notación de éstos va a permitir que muchos practicantes de la programación salvaje (o, mejor dicho, de la programación heroica) se vayan convenciendo paulatinamente de la conveniencia de modelar las bases de datos antes de generarlas. Por una parte, les va a permitir descubrir muchos problemas que no son evidentes cuando se escribe o se revisa el texto de los esquemas (o de las proposiciones CREATE TABLE); y, por otra parte, estamos seguros que les va a agradar mucho el poder ahorrarse la escritura de este código.

El OMG y los proponentes de UML buscaron también cubrir otros aspectos. Además de obviarnos el tener que decidir entre varias formas de representación, UML nos proporciona un lenguaje visual de modelado, y nos permite aprovechar componentes, plantillas y patrones a partir de soluciones exitosas que ya han sido probadas. En las notas bibliográficas, al final del presente artículo, se podrá encontrar la referencia a una página de Internet dedicada específicamente a estas plantillas y patrones, que representan una de las técnicas más prometedoras para facilitar el desarrollo de sistemas.

Para mantener su característica de estándar abierto, UML es independiente de los lenguajes, de las bases de datos y de la marca del software y del equipo de cómputo. Deliberadamente, los proponentes de este lenguaje unificado lo han dejado únicamente como un medio para especificar, visualizar y documentar los artefactos del desarrollo e implantación de sistemas. Es importante hacer hincapié en el

hecho de que UML no es un método de análisis y diseño, ni pretende representar un proceso para guiar las actividades de un desarrollo. Los usuarios de distintos métodos podrán mantener su forma actual de trabajo mediante adaptar sus diagramas de acuerdo a los símbolos utilizados en UML. En la mayor parte de los casos esta adaptación se va a poder realizar con facilidad.

La independencia de UML con respecto a los métodos de análisis y diseño nos beneficia también por otro motivo. Podemos afirmar que una de las razones por las que han proliferado los métodos de análisis y diseño es que no existe una secuencia rígida de actividades que sea aplicable a todos los tipos de desarrollo; es decir, que los distintos proyectos requieren de distintas formas de solución. UML no pretende resolver este problema mediante plantear un *proceso* único a seguir, sino que sólo contempla la estandarización a nivel de símbolos y de notación. Este enfoque tiene la ventaja de lograr una gran flexibilidad en cuanto a aprovechar técnicas procedentes de diversos autores.

Toda persona familiarizada con la diagramación va a encontrar que el nuevo lenguaje unificado ha incorporado muchas de las mejores técnicas de la industria, gracias a las múltiples aportaciones de quienes hicieron posible la propuesta de estandarización. Por ejemplo, UML nos permite modelar sistemas concurrentes y distribuidos, con lo cual se pueden representar características que han sido incorporadas muy recientemente a ciertos lenguajes de programación, como es el caso de Java.

Para poder llegar a un lenguaje estándar, los proponentes de UML buscaron definir con precisión el significado de los símbolos, así como las formas de usarlos y relacionarlos. Fue necesario que se alcanzara un amplio acuerdo en torno a cuáles deben ser los artefactos a usar durante el análisis y el diseño; es decir, sobre cuáles deben ser los componentes de los modelos. Esta es precisamente la función del metamodelo, como veremos en los siguientes párrafos.

Qué es el metamodelo

Para el grupo del OMG encargado de establecer las bases del concurso y de evaluar las propuestas (conocido como *OA&D Task Force*) el problema no consistía únicamente en solicitar una serie de normas para crear diagramas, sino que estos diagramas y sus componentes deberían ser compatibles con los servicios necesarios para transportarlos de una herramienta de diagramación a otra (lo cual podría significar también el transportar diagramas de una plataforma a otra).

Una de las condiciones del concurso consistió en que el lenguaje de modelado debería ser coherente con el servicio necesario para proporcionar y transportar los componentes de los modelos. Este servicio es conocido como Meta-Object Facilty (MOF).

De hecho, los concursos para la definición del UML y del MOF se realizaron en paralelo. Para empatar ambos, los proponentes recurrieron a una arquitectura de cuatro capas:

Un **meta-metamodelo** que define el lenguaje para la siguiente capa, que es el metamodelo Suena complicado, pero la idea es más simple de lo que parece: esta capa consiste de un modelo compacto que sirve de cimentación para el conjunto de la arquitectura. El meta-metamodelo es común al MOF y a UML, lo que asegura que compartan una base común. El meta-metamodelo asegura, por ejemplo, que los tipos de datos manejados por el lenguaje de modelado y el servicio MOF sean totalmente compatibles.

El **metamodelo**, que es una instancia (es decir, una expresión particular) del meta-metamodelo. Esta es la capa donde se define el lenguaje que sirve para especificar los modelos que vamos a crear los usuarios de UML. En otras palabras, el metamodelo sirve para describir los elementos que van a componer nuestros diagramas. Esta capa es propia de UML (ya que el MOF tiene su propio metamodelo), y es aquí donde se definen los objetos del lenguaje unificado: *Clase*, *Atributo, TipoDato...*

De acuerdo a esta arquitectura, cuando los usuarios creamos el objeto *Empleado* en un diagrama estamos creando una instancia del metaobjeto *Clase*; es decir, que estamos usando *Clase* como si fuera el molde para producir una expresión concreta (con características propias) llamada *Empleado*. De igual forma, si al objeto *Empleado* le añadimos el atributo *Categoría*, en realidad estamos instanciando la metaclase *Atributo*.

El **modelo** es una instancia del metamodelo, y es lo que los usuarios de UML vamos a crear. Cuando creamos un modelo estamos definiendo un lenguaje para describir el área que estamos analizando o el sistema que estamos diseñando. Cuando creamos los objetos *Empleado*, *Departamento* y *Categoría* estamos creando objetos que pertenecen a esta capa, y que servirán a su vez de moldes para los datos que vamos a introducir, manipular, almacenar y procesar en nuestras aplicaciones.

Los **objetos del usuario** (o, si así queremos llamarlos, los **datos del usuario**) son los datos y objetos que describen el área o dominio a los que está dedicada la aplicación. Por ejemplo, los nombres y características de personas almacenados en el objeto *Empleado* son elementos de esta capa.

Una de las principales ventajas de este enfoque es que cada uno de los objetos puede ser validado contra su respectivo metaobjeto, en la capa inmediata superior. Por ejemplo, la clase *Empleado* debe corresponder a las características y especificaciones de *Clase* en el metamodelo.

Entremos, pues, a ver el metamodelo mismo.

Los tres paquetes principales

Como podemos ver en la Figura 1, el metamodelo de UML está dividido en tres paquetes, que contienen todos los elementos de modelado y los tipos de diagramas que conforman el lenguaje unificado.

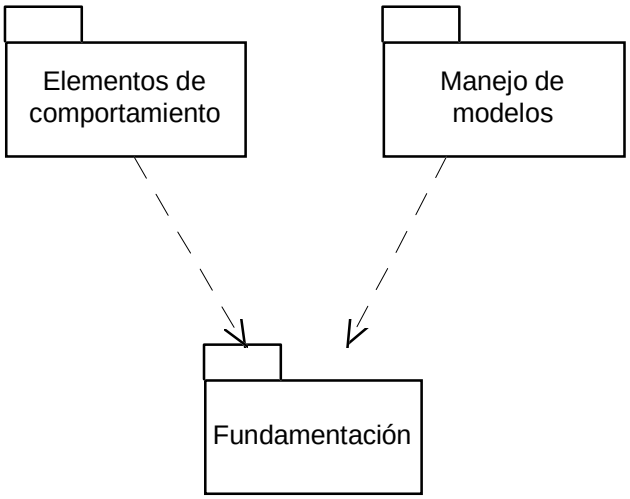

Figura 1. Paquetes del nivel superior (*UML Semantics*, p. 8.)

El paquete de Elementos de comportamiento (*Behavioral Elements Package*) define los elementos necesarios para representar la dinámica de un sistema, como son los casos de utilización (*use cases*), los diagramas de interacción, los diagramas de actividades y los diagramas de estado.

El paquete de Manejo de modelos (*Model Management package*) especifica cómo organizar los elementos de UML en modelos, paquetes y sistemas.

Más adelante vamos a abordar el paquete de Fundamentación (*Foundation package*), y vamos a ver algunos de sus elementos con más detalle. A este paquete pertenecen, por ejemplo, los elementos que componen los diagramas de clases, que son utilizados para representar la estructura de un sistema.

En la Figura 1 podemos apreciar que se ha utilizado un tipo de diagrama de UML, llamado “diagrama de paquetes”, para poder agrupar a sus componentes. Los mismos tipos de diagramas que vamos a utilizar para modelar sistemas son los que fueron usados para representar el metamodelo.

Los diagramas de paquetes sirven para agrupar los diferentes componentes de un sistema. Los iconos en forma de carpeta representan los paquetes (que, a su vez, pueden contener subpaquetes).

Las flechas no representan las interacciones entre los paquetes, sino las *dependencias* entre éstos. Lo que nos está diciendo el diagrama en la Figura 1 es que si se llegaran a realizar cambios en el paquete de Fundamentación va a ser necesario revisar la repercusión de estos cambios en los paquetes de Elementos de comportamiento y de Manejo de modelos.

Un vistazo al paquete de Fundamentación

El paquete de Fundamentación define la infraestructura de UML. Por ejemplo, ahí se define que existen varios tipos de *ElementoDelModelo*. En la Figura 2 podemos apreciar una parte de la "columna vertebral" de este paquete. (Precavemos al lector que, para simplificar la explicación, no estamos incluyendo todos los componentes y relaciones de esta "columna vertebral".)

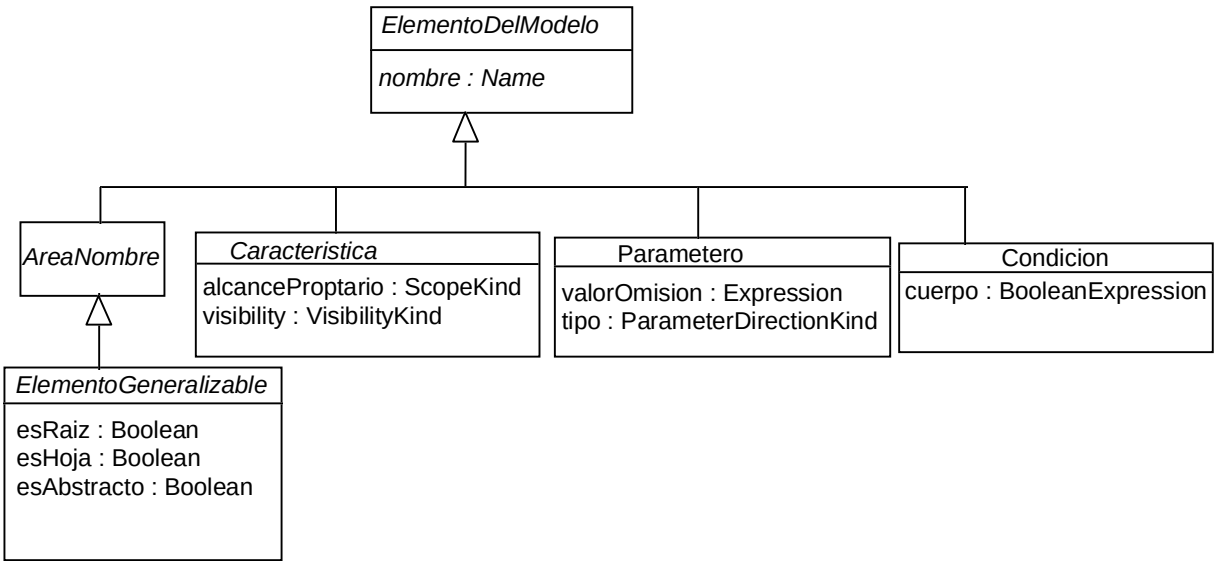

Figura 2. Tipos de *ElementoDelModelo.* (*UML Semantics*, p. 16.)

En la Figura 2 podemos apreciar que el modelo de un sistema puede tener varios tipos de elementos:

AreaNombre (*Namespace*), que son aquellas áreas o espacios en un modelo donde no puede haber más de un elemento con el mismo nombre. Por ejemplo, si hay un objeto que se llama *Empleado* en un diagrama, podemos decir que en ese ámbito no deberá usarse *Empleado* como nombre de otro elemento.

Caracteristica (*Feature*), que son propiedades que se encuentran encapsuladas dentro de otro elemento. Este es el caso de los atributos y operaciones que podría tener *Empleado*, tales como el número de empleado o el puesto que desempeña.

Parametro, que representa aquellas variables que pueden ser modificadas, pasadas de un elemento a otro, o regresadas. Los parámetros pueden tener un nombre, un tipo y una dirección.

Condicion (*Constraint*), que son afirmaciones para especificar condiciones o restricciones. Por ejemplo, mediante la condición {ordered} se puede establecer que las instancias de *Empleado* deben estar ordenadas de acuerdo al número del empleado.

ElementoGeneralizable es un elemento de tipo *AreaNombre* que puede participar en una relación de generalización. Las clases de objetos como *Empleado* pueden participar en relaciones de generalización, ya que en nuestros modelos puede haber distintos tipos de empleados.

El tipo de diagrama utilizado en la Figura 2 se llama diagrama de clases. Estos son los diagramas de UML donde se representan los tipos de objetos y las relaciones estáticas que hay entre ellos. Las cajas representan las clases de objetos (llamadas simplemente “clases”), y las relaciones entre éstas se representan mediante líneas. En la parte superior de las cajas se encuentra el nombre de la clase (*ElementoDelModelo, Característica*, Parametro...) y en la división inferior se encuentran sus atributos. Como se puede observar, el formato para declarar los atributos es: *nombreAtributo* : *tipodato.*

Como se trata del metamodelo, las clases de la Figura 2 son consideradas “metaclases” (que son los moldes con los que vamos a formar los elementos de nuestros modelos). Me he tomado la libertad de traducir al castellano los nombres de estas metaclases y sus atributos, pero he dejado los tipos de datos en inglés.

Las cabezas de flecha sin rellenar que apuntan hacia las cajas son el símbolo de generalización en UML. Quien no esté familiarizado con la orientación a objetos puede leer este símbolo simplemente como “es un tipo de...”. En la Figura 2, por ejemplo, *Característica* es un tipo de *ElementoDelModelo*, y Parametro es un tipo de *ElementoDelModelo...*

El lector se preguntará por qué los nombres de las metaclases *AreaNombre, Caracteristica* y *ElementoGeneralizable* se encuentran en cursivas, mientras que Parametro y Condicion se encuentran en letra normal. Esto se debe a que en UML los nombres de las clases abstractas se escriben en letra cursiva, mientras que los de las clases concretas se escriben en letra normal. Las clases abstractas son tipos de objetos que no tienen instancias concretas en un sistema. Por ejemplo, en los modelos que creemos mediante UML no nos vamos a encontrar con elementos que representen directamente a la metaclase *ElementoDelModelo* (que es una clase abstracta), pero sí vamos a tener elementos que representan directamente a Parametro, a Condición y a otras clases concretas que aparecen en la Figura 3.

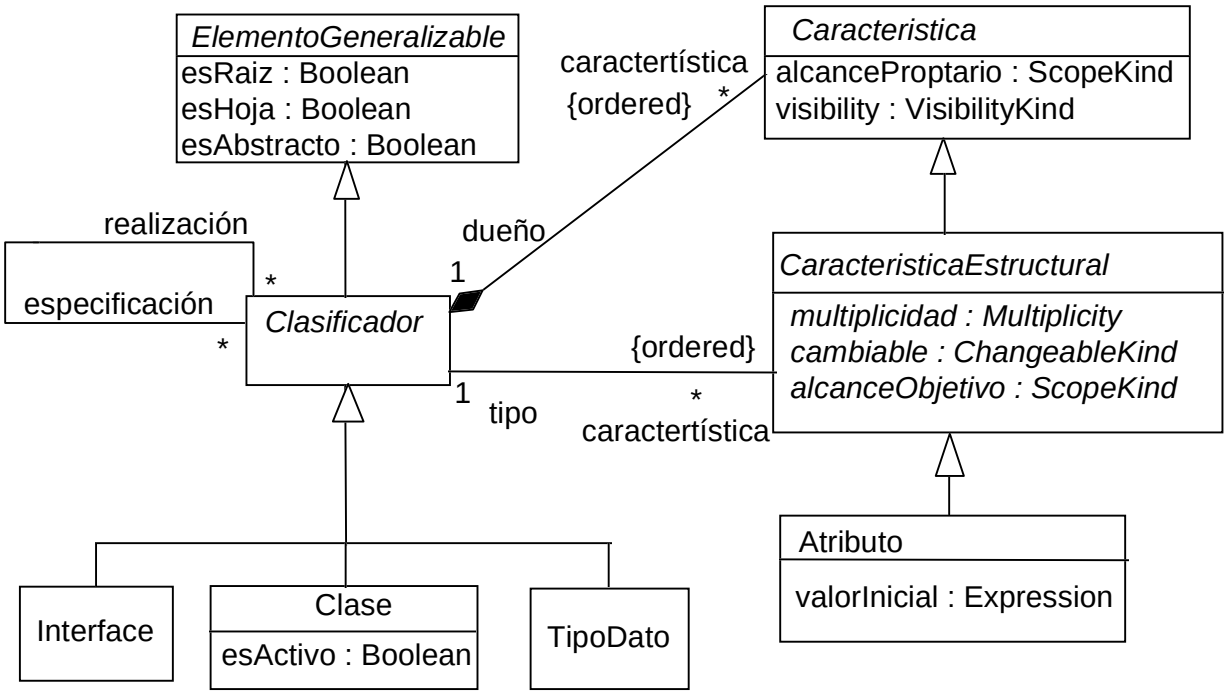

Figura 3. Tipos de *ElementoGeneralizable.* (*UML Semantics*, p. 16.)

Puede observarse que hay una subclase de *ElementoGeneralizable* que se llama *Clasificador*. Nos podemos preguntar por qué la generalización no se hizo directamente hacia *ElementoGeneralizable* (a partir de Interface, Clase y TipoDato) sin tener que pasar por *Clasificador*. La razón es que hay varias

relaciones que ligan a *Clasificador* con otras clases, y que aplican tanto a Interface, como a Clase y a TipoDato. La clase abstracta *Clasificador* nos ahorra el tener que dibujar todas estas relaciones hacia cada una de sus subclases.

Una de estas relaciones tiene un rombo negro en uno de sus extremos, lo que indica que un C*lasificador* puede estar compuesto de una o varias instancias de *Caracteristica*. En este caso, *Clasificador* juega el papel de dueño de las características, mismas que están sujetas a la condición {ordered}, lo que significa que las instancias de *Caracteristica* deben estar en un orden determinado.

En el diagrama también se aprecia que un *Clasificador* puede estar relacionado con una *CaracteristicaEstructural*, o con varias, que *CaracteristicaEstructural* es un tipo de *Caracteristica* y que Atributo es un tipo de *CaracteristicaEstructural.*

¿Debemos preocuparnos por el metamodelo?

En los párrafos anteriores sólo hemos cubierto una pequeña (pero significativa) parte del metamodelo de UML. Esperamos haber explicado dos aspectos que nos propusimos como objetivo al escribir este artículo: primero, presentar un panorama de la forma en que está fundamentado el nuevo lenguaje de modelado; y, segundo, presentar ejemplos de diagramas creados con UML. Con esto no hemos querido dar la impresión que los usuarios debemos preocuparnos de aprender el metamodelo, o que requiramos manejar o modificar la fundamentación del lenguaje unificado. Más bien, nuestra intención ha sido la de presentar a UML como un estándar robusto. Todo lo que esperamos, como usuarios, es que esta infraestructura funcione eficiente y silenciosamente (y es por eso que su encanto debe realmente ser discreto). En forma análoga, podemos gozar de los servicios de un edificio sin necesariamente conocer la forma en que su cimentación y su estructura fueron concebidas, calculadas o armadas.

La extensión del presente artículo no nos permite abordar ejemplos de utilización de UML, pero en las notas bibliográficas podrán encontrarse recomendaciones de textos para quienes estén interesado en el aspecto práctico. Por otra parte, si algún lector deseara conocer más sobre el metamodelo mismo, sugerimos el texto *UML Semantics* (aunque debemos precaver que dicho documento es de difícil lectura).

El concepto de *metamodelado* puede ser muy útil a quienes estén buscando soluciones lógicas para problemas de interoperabilidad, o a quienes deseen comparar diferentes herramientas CASE. Sobre este tema recomendamos la página de Internet sobre metamodelos mencionada en las notas bibliográficas.

El modelado no sustituye la construcción de sistemas

Debemos tener siempre presente el objetivo central del desarrollo de sistemas: lo que esperan los clientes que solicitan un solución informática es llegar a contar con un sistema confiable y amigable. El modelado es sólo un medio para alcanzar esta meta, y no un fin en si mismo. Para un equipo de trabajo capaz de adaptarlo a las circunstancias, UML va a ser de gran ayuda; pero los modelos creados no van a sustituir la labor de programación y aplicación de pruebas. Por el momento UML es sólo un lenguaje de modelado, y no pretende ser un lenguaje para la programación mediante símbolos visuales. Existen muchas operaciones, como es el caso de JOIN, que es posible expresarlas más claramente en un lenguaje de programación que en un diagrama.

¿Qué tanto va a contribuir UML para establecer una verdadera ingeniería de software, que nos permita construir sistemas mediante un proceso controlable, predecible y repetible? La respuesta a esta pregunta depende de que se demuestre, en casos reales, que el uso del modelado de sistemas nos va a permitir hacer las cosas mejor, dentro de costos razonables, y a tiempo.

Notas bibliográficas y direcciones en la red

Es muy recomendable la serie de artículos “Object Orientation: Making the Transition”, de Meilir Page-Jones, que aparece en la revista Eiffel Liberty en **http://www.elj.com/elj**. Page-Jones expone en esta serie una versión actualizada de los temas tratados en la conferencia videograbada mencionada al principio del presente artículo. No se trata de la usual alabanza a la orientación a objetos, sino de una exposición crítica tanto de las oportunidades como de las limitaciones y problemas asociados a esta tecnología.

Sobre plantillas y patrones, consúltese **http://hillside.net/patterns**. Vale mucho la pena leer la introducción de Brad Appleton, titulada “Patterns and Software: Essential Concepts and Terminology”, que se encuentra bajo el apartado “About Patterns”.

El libro de Martin Fowler y Kendall Scott, *UML Distilled* (Addisson Wesley, Reading, Mass., 1997) es una excelente introducción y guía práctica para iniciarse en el uso del lenguaje unificado.

Grady Booch, James Rumbaugh e Ivar Jacobson han terminado de elaborar *Unified Modeling Language User Guide*, que ha sido publicado por Addison Wesley. Se encuentran terminando dos libros más: *Unified Modeling Language Reference Manual* y *The Objectory Software Development Process*, que Addison Wesley estima que serán publicados en el verano de 1998.

Sobre métodos de análisis y diseño se pueden consultar las siguientes fuentes:

Jacobson, Christerson, Jonsson, Overgaard*: Object Oriented Software Engineering: A Use Case Driven Approach* (Addison-Wesley and ACM Press, Reading, Mass., 1992).

James Martin y James Odell: *Métodos orientados a objetos: consideraciones prácticas* (Prentice-Hall, México, 1997).

Philipp Schneider ha dejado disponible en la página de Internet del Interkantonales Technikum Rapperswil (ITR), de Suiza, una guía en inglés del método de Booch. Véase **http://www.itr.ch/courses/case/BoochReference**

Sobre el concepto de metamodelado se puede consultar **http://www.metamodel.com**

El Object Management Group ha difundido un texto introductorio donde explica la historia y las razones por las cuales adoptó el UML. Este texto se encuentra en **http://www.omg.org/news/pr97/umlprimer.htm**

La descripción de UML y su metamodelo la hemos tomado de los documentos siguientes, presentados por Rational Software, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing, IntelliCorp, i-Logix, IBM, ObjecTime, Platinum Technology, Ptech, Taskon, Reich Technologies y Softeam:

UML Proposal to the OMG, version 1.1, 1 September 1997.

UML Summary, version 1.1, 1 September 1997.

UML Semantics, version 1.1, 1 September 1997.

UML Notation Guide, version 1.1, 1 September 1997.

Las actualizaciones más recientes a estos documentos se encuentran disponibles en: **http://www.omg.org/library/schedule/Technology_Adoptions.htm#tbl_UML_Specification**