# Una propuesta para incorporar más semántica de los modelos al código generado

Sonia Pérez Lovelle, Julio C. Cue Galindo, Alexei Hernández Perenzuela,
Andry Arredondo López, Luis R. Recio Nápoles, Frances Carnero González

***Resumen*—Actualmente hay un amplio uso del paradigma Model Driven Architecture (MDA) para la generación de código a partir de modelos, pues esto garantiza menores tiempos de desarrollo y de puesta a punto. Los modelos creados a partir de los diagramas del Lenguaje Unificado de Modelado (UML) son de amplia utilización teniendo en cuenta que se trata de un estándar y además, la gran cantidad de herramientas de modelado que existen para ello. Cada diagrama de UML es un punto de vista diferente del sistema modelado, pero cada uno de estos, tiene su sintaxis y su semántica y aporta información para el código resultante. La forma de intercambiar estos diagramas entre las diferentes herramientas es a través del uso de ficheros XMI (XML Metadata Interchange). XMI es un estándar, sin embargo, no todas las herramientas de modelado tienen las opciones de importar / exportar para este formato y las que lo hacen, no permiten la total interoperabilidad entre herramientas, debido a que usan sus propias estructuras. En este trabajo se aborda la semántica del diagrama de clases y cómo se refleja esta en el código generado por la herramienta AndroMDA, precisando los aspectos que pueden mejorarse en función de la semántica de UML, a partir de la modificación de sus cartuchos.**

***Palabras clave*—AndroMDA, diagrama de clases, MDA, semántica de UML, XMI.**

# A Proposal to Incorporate More Semantics from Models into Generated Code

***Abstract*—Currently, there is a widely used paradigm called Model Driven Architecture (MDA) for code generation from models, because this ensures shorter development times. The models created from the diagrams of Unified Modeling Language (UML) are widely used, considering that it is standard and a large number of modeling tools exists for it. Each UML diagram is a different view of the modeled system, but each of them has its syntax and semantics and each of these elements provides information for the resulting code. These diagrams are exchanged between different tools using XMI files (XML Metadata Interchange). XMI is a standard; however, not all modeling tools have options to import / export to this format and they do not allow full interoperability between tools, because they use their own structures. This paper addresses the semantics of class diagram and how it is reflected in the code generated by the AndroMDA tool, specifying the aspects for improvement based on the semantics of UML through modification of their cartridges.**

***Index Terms*—AndroMDA, class diagram, MDA, UML semantics, XMI.**

Manuscrito recibido el 20 de marzo de 2013; aceptado para la publicación el 29 de julio del 2013; versión final 13 de junio 2014.

Sonia Pérez Lovelle está con la Facultad de Ingeniería Informática del Instituto Superior Politécnico José Antonio Echeverría, Cuba (correo: sperezl@ceis.cujae.edu.cu).

Julio C. Cue Galindo, Alexei Hernández Perenzuela, Andry Arredondo López y Luis R. Recio Nápoles están con el Centro de Desarrollo de Aplicaciones de Tecnologías y Sistemas, Cuba (correo: {julio.cue, alexei.hernandez, andry.arredondo, luis.recio}@datys.cu).

Frances Carnero González está con el Grupo de Electrónica para el Turismo, Cuba (correo: frances@get.mintur.cu).

## I. INTRODUCCIÓN

La arquitectura dirigida por modelos (MDA, por sus siglas en Inglés), propone un proceso de desarrollo basado en la realización y transformación de modelos. MDA ha permitido minimizar los tiempos y las dificultades al desarrollar aplicaciones con alto nivel de complejidad.

AndroMDA es una de las herramientas que implementan el paradigma MDA. Su ambiente de trabajo se basa en la entrada de un modelo especificado en UML [1] (Unified Modeling Language, por sus siglas en Inglés) y exportado en formato XMI (XML Metadata Interchange, por sus siglas en Inglés), por una herramienta de modelado y genera código para las tecnologías que funcionan dentro del ámbito de la arquitectura de varias capas propuesta por ella [2].

El diagrama central en este proceso de generación de código es el diagrama de clases, en el cual la sintaxis es alterada durante el proceso de diseño, por ejemplo, cuando se sugiere que para la generación de código, en el modelado se deben describir todos los atributos pasivos de una clase con visibilidad pública, en contra de todas las buenas prácticas, pero el código resultante responde a las exigencias de un correcto diseño. Existen elementos sintácticos, como la navegabilidad en determinadas relaciones que tienen una alta carga semántica y en este caso, se dejan de generar elementos. Sin embargo, para el caso de la semántica asociada a los diferentes tipos de relaciones presentes en el diagrama, no siempre es plasmada en el código resultante, lo cual provoca que existan elementos que deben ser adicionados al código una vez obtenido y por tanto, implica dedicar tiempo a tareas que en teoría están resueltas.

En las secciones II, III y IV se trata lo concerniente a los diagramas de clases de UML, el tratamiento de las

herramientas y a la semántica del diagrama de clases, respectivamente, para en la sección IV abordar la interpretación que hace AndroMDA de la semántica del diagrama de clases de UML para poder justificar en la sección VI las modificaciones necesarias en los cartuchos de AndroMDA. Finalmente, en la sección VII se muestran algunas conclusiones y el trabajo futuro para complementar la solución actual.

## II. Diagrama de clases de UML

El diagrama de clases es uno de los catorce diagramas que posee UML para la modelación de un sistema. Se trata de un diagrama estructural, que muestra cada una de las clases con sus métodos y atributos, así como sus relaciones con otras clases.

Su mayor carga sintáctica está dada en las clases, mientras que la semántica se encuentra fundamentalmente en las relaciones que se establecen entre las clases.

Los tipos de interrelaciones que se pueden establecer entre las clases son relaciones de generalización/especialización, asociación, agregación y composición.

Generalización / especialización: Una generalización es una restricción taxonómica entre dos clases. Esta restricción especializa una clase general (clase padre) en una clase más específica (clase hija). Las especializaciones y generalizaciones son dos puntos de vista del mismo concepto. Las relaciones de generalización forman una jerarquía sobre un conjunto de clases. Una jerarquía de generalización induce una relación de subconjunto en el dominio semántico de clases [3].

Asociación: Es un enlace entre instancias de los tipos asociados. Un enlace es una tupla, con un valor para cada final de la asociación, donde cada valor es una instancia del tipo del final [1].

Agregación: Modela la relación *conoce-un*. La clase que agrega o conoce, no tiene que construir a la agregada o conocida, ya que esta existe fuera de su contexto. En UML esta relación se modela a través de un rombo no relleno.

Composición: Modela la relación *tiene-un*. Está formada en sus extremos por una relación *"todo/partes"*. Cuando se construye la instancia de la clase que representa el extremo *"todo",* deben construirse también las instancias de la "parte", o sea, su constructor debe invocar al constructor de las instancias miembros para que se construyan como corresponde [4].

Existe además la Clase de Asociación, (AssociationClass). Según [1], puede verse como una clase que a la vez es una asociación, o una asociación que es también una clase. Este elemento se origina a partir de una relación que exista entre clases en la que ciertas características son propias de la relación en sí y no de alguna de las clases implicadas en la relación.

## III. Herramientas de modelado UML

Las herramientas de modelado de UML existen casi desde la propia aparición de este lenguaje y han aumentado notablemente en número, existiendo en la actualidad una gran variedad de ellas, tanto propietarias, como libres y de código abierto.

Sus primeras versiones solo permitían el modelado de diagramas de UML, con formatos propios por lo que no se podía intercambiar información entre las diferentes herramientas.

Las herramientas salvan y procesan la sintaxis de los diagramas de UML, más cercana a los elementos gráficos, sin embargo, no siempre realizan validaciones de estos, lo cual tiene implicaciones negativas en la interpretación de sus significado, pero que no tienen mayores implicaciones mientras solo las herramientas se dedicaban a salvar y recuperar, pero que se convierten en claves cuando se trata de generar código.

A partir de la aparición del formato XMI como un estándar, se abren las puertas para la interoperabilidad, no solo entre herramientas de modelado, aunque esto hubiera sido ideal y algunas de estas herramientas incorporan la importación y exportación usando este nuevo formato, pero de manera general no hay un respeto total del estándar y por tanto, se pierde en gran medida la deseada interoperabilidad.

Algunos estudios realizados [5], [6] indican no solo que no existe una compatibilidad total entre las herramientas de mayor uso, sino que en algunos casos ni tan siquiera se logra importar lo que antes fue exportado por una herramienta.

Por esta razón, AndroMDA exponente del paradigma MDA, declara que su mayor "compatibilidad" o "entendimiento" es con el fichero XMI generado por la herramienta Enterprise Architect, lo cual es un reconocimiento de que estos ficheros no se ajustan a lo establecido en el estándar.

## IV. Semántica del diagrama de clases de UML

A pesar de que algunos autores sostienen que UML tiene una definición semántica flexible [7], [8] o no formalmente definida [9], [10], [11], que no permite la generación de código o modelos ejecutables [12], es posible a partir de la propia sintaxis de los diagramas establecer diferencias semánticas para elementos del diagrama con diferente sintaxis como se describe a continuación de acuerdo con su significado, teniendo en cuenta que los formalismos pueden ser punto de partida para tras procesos de refinamiento, llegar a la obtención del código resultante [13].

La clase como elemento fundamental en este diagrama, da significado diferente a los atributos pasivos y activos, así como a la visibilidad de estos.

Las asociaciones entre clases permiten obtener información sobre visibilidad, navegabilidad y cardinalidad, para aquellos casos en que aparezcan descripciones en sus extremos.

Para el caso de las relaciones de generalización / especialización que representan relaciones jerárquicas de herencia, la

información semántica está asociada con la posibilidad de poder establecer si se trata de herencia solapada o disjunta por un lado y si se trata de herencia parcial o total.

Las relaciones de agregación y composición, son relaciones del tipo Todo/Parte, que sintácticamente solo se diferencian por el color del rombo, sin embargo, este tipo de rombo indica si las ocurrencias del tipo Parte pueden existir independientemente de las ocurrencias del tipo Todo y por tanto, deben ser tratadas de manera diferente.

La clase de asociación que surge de la asociación entre dos clases, que aporta información tanto en la clase como en la asociación atendiendo a la semántica o significado de ambos elementos.

## V. Interpretación de AndroMDA del diagrama de clases

La herramienta AndroMDA recibe un fichero en formato XMI donde deben aparecer de manera serializada todos los elementos de un diagrama de clases, sin embargo este fichero generado por una herramienta CASE no tiene en cuenta todos los elementos y en ocasiones, las propias herramientas no son capaces de reflejar todas las posibilidades que brinda UML.

Producto de esto, en el proceso de interpretación hacia el código generado dejan de tenerse en cuenta elementos que aparecen en un diagrama de clases con un significado preciso y que por lo tanto tienen una función en el resultado final.

Aunque los principales problemas se encuentran en el código generado a partir de las relaciones, en el caso de las clases, existen dos situaciones que deben ser consideradas. Una es la clase de asociación para la cual no se genera nada relacionado con la clase, solo con la asociación y la otra tiene que ver con las superclases diferentes de la primera en una herencia múltiple, que no son tenidas en cuenta en el código resultante.

Esta última situación está dada por el hecho de que el lenguaje Java no permite este tipo de herencia, sin embargo, hay estrategias para representarla, como la propuesta en [14], en la que se hace uso de interfaces.

En el caso de las relaciones, al no darle tratamiento a la herencia múltiple, no se tienen en cuenta los atributos que se le pueden adjudicar a esta desde el punto de vista del solapamiento y si es o no total.

Las relaciones de agregación y composición son interpretadas de la misma forma, a pesar de tener un significado diferente y en consecuencia, a los efectos del código que se genera no existen diferencias y no es importante especificar uno u otro tipo de relación.

## VI. Modificación de cartuchos de AndroMDA

AndroMDA no se restringe solo a generar código para las tecnologías o plataformas que provee por defecto. Debido a la arquitectura basada en cartuchos con la que se ha concebido esta herramienta, es posible modificar, crear e incluir cartuchos para generar código para cualquier plataforma [15].

Para lograr dar solución a los problemas planteados, se modificaron los cartuchos Hibernate, Java y Spring tanto para la versión 1.x de UML [16], como para la versión 2.0 [17].

Los cartuchos procesan los elementos del modelo utilizando archivos de plantillas en el lenguaje Velocity, definidas dentro del descriptor del cartucho [2]. En dichas plantillas se detalla el código que se desea generar y se agregan además los valores de determinadas variables o atributos contenidos en las clases metafachadas [15].

AndroMDA presenta como elemento principal un "motor" o "*core*", encargado de guiar el funcionamiento del proceso de generación.

El motor de la herramienta detecta los cartuchos utilizados. La información de estas dependencias se encuentra en el fichero descriptor del proyecto, ***pom.xml*** situado en la raíz del proyecto. Al leer este fichero, AndroMDA es capaz de definir cuáles serán los cartuchos a utilizar y su ubicación dentro de un repositorio establecido [15], [18].

A partir de los problemas detectados y enunciados anteriormente, se modificaron algunas plantillas en los cartuchos y se crearon otras, como se indica a continuación.

Para el tratamiento de la herencia múltiple se realizaron cambios en la plantilla *Interface.vsl* del cartucho *Java* con el objetivo de generar los atributos de las interfaces y en la plantilla *hibernate.hbm.xml.vm* del cartucho *Hibernate* para garantizar incluir estos atributos en la base de datos correspondiente.

En todos los casos, la implementación se realizó teniendo en cuenta las posibles colisiones que se pueden originar a partir de tener más de una superclase, lo cual fue resuelto de acuerdo con lo expresado en [19].

Para la implementación de la clase de asociación se modificó la plantilla ***HibernateEntity.vsl*** para generar este tipo de clase y sus relaciones y se crearon dos plantillas en el cartucho ***Hibernate***: ***HibernateAssociationClassEntity.vsl*** e ***HibernateAssociationClassEntityImpl.vsl***.

Además, para garantizar el acceso a los datos de dicha clase de asociación, se crearon, tres plantillas, en el cartucho ***Spring, SpringAssociationClassDao.vsl***, ***SpringHibernateAssociationClassDaoBase.vsl*** y ***SpringHibernateAssociationClassDaoImpl.vsl***.

Por último, para lograr la diferenciación en el código generado de las relaciones de composición y agregación, se realizaron modificaciones en la plantilla ***HibernateEntity.vsl*** del cartucho ***Hibernate.*** El código modificado hace posible que en el constructor de la clase que representa el extremo "*todo*" de la relación de composición, se cree la instancia de la clase que representa el extremo "*parte*", representando así su significado real.

## VII. Conclusiones

Las herramientas de modelado de UML, de manera general, no tienen en cuenta el significado asociado a cada uno de los elementos gráficos que aparecen en los diagramas de UML y

no siguen el estándar XMI, lo cual provoca que exista una brecha semántica entre el diagrama y su serialización en el fichero XMI, lo que provoca que en el código generado por la herramienta AndroMDA no se tengan en cuenta las especificidades de cada tipo de elemento, sin embargo, ha quedado demostrado que es posible modificar o extender los cartuchos y con ello lograr incorporar mayor semántica al código generado, lo cual redunda en menor tiempo dedicado a mejorar el código generado de manera automática y por consiguiente en acortar en alguna medida el tiempo de desarrollo.

Como trabajo futuro, se trabaja en la incorporación de los diagramas de transición de estados y de actividad.

## REFERENCIAS


[1] OMG. *Unified Modeling Language (UML) Specification: Superstructure version 2.0*. 5/7/2004; http://www.uml.org

[2] AndroMDA *Home Page*; http://www.andromda.org.

[3] P. Nieto Soler, "Redefinición de Asociaciones en UML: Semántica y Utilización", Tesis de Maestría, Dep. de Lenguajes y Sistemas Informáticos. Universidad Politécnica de Cataluña, España, 2008.

[4] IBM. *Rational Unified Process (RUP)*; http://www-01.ibm.com/software/awdtools/rup/

[5] A. Bordón, L. García, D. O. Hernández Darién, "ACGTool: Herramienta de soporte a la instancia de la Arquitectura de Componentes Genéricos usando UML", Trabajo de Diploma, Instituto Superior Politécnico José Antonio Echeverría, La Habana, Cuba, 2007.

[6] N. Fuentes Ramírez, "Sistema automatizado para la conversión de ficheros XMI de herramientas de modelado UML", Tesis de Maestría en Informática Aplicada. Fac. Ing. Informática. Inst. Sup. Pol. José A. Echeverría. La Habana, Cuba, 2008.

[7] M. Björdanker, C. Kobryn, "Architecting systems with UML 2.0". IEEE Software. July / August, 2003.

[8] D. Thomas, "UML – Unified or Universal Modeling Language?". *Journal of Object Technology*, vol. 2, no. 1, 2003.

[9] D. Jäger, A. Schleicher, B. Westfechtel, "Using UML for Software Process Modeling", in: O. Nierstrasz, M. Lemoine, (eds.), *ESEC/FSE'99*, *Lecture Notes in Computer Science*, vol. 1687, 1999

[10] R. Heckel, J. Kuster, G. Taentzer, "Towards Automatic Translation of UML Models into Semantic Domains", *APPLIGRAPH Workshop on Applied Graph Transformation (AGT 2002)*, Grenoble, France, 2002

[11] R. B. France, S. Ghosh, T. Dinh-Trong, "Model-Driven Development Using UML 2.0: Promises and Pitfalls", IEEE Computer Society, 2006

[12] J. Hogg, "Brass Bubbles: An overview of UML 2.0 (and MDA)", *Fourth Workshop on UML for Enterprise Applications: Delivering the Promise of MDA*, IBM Software Group, OMG, Junio 2003.

[13] M. Egea González, "Una semántica formal ejecutable para OCL con aplicaciones al análisis y a la validación de modelos", Ph.D. tesis, Dept. Sist. Inf. y Computación. Univ. Complutense de Madrid, 2008

[14] S. Holzner, "La Biblia de Java 2 [Multimedia]", Madrid: Anaya Multimedia, 2000

[15] D. Pagés Chacón, "Cartucho de AndroMDA para JSF Interpretando Nueva Estrategia de Modelado", Tesis de Maestría en Informática Aplicada. Fac. Ing. Informática. Inst. Sup. Pol. José A. Echeverría, La Habana, Cuba, noviembre 2010

[16] A. Arredondo López, L. R. Recio Nápoles, "Modificación de cartuchos de AndroMDA para incluir más semántica del diagrama de clases de UML 1.4", Trabajo de Diploma, Instituto Superior Politécnico José Antonio Echeverría, La Habana, junio 2012

[17] J. C. Cue Galindo, A. Hernández Perenzuela, "Modificación de cartuchos de AndroMDA para incluir más semántica del diagrama de clases de UML 2.0". Trabajo de Diploma, Instituto Superior Politécnico José Antonio Echeverría, La Habana, junio 2012

[18] E. Hernández Lee, "Extensión al cartucho bpm4struts de AndroMDA para la generación de componentes de prueba para entornos especializados en struts 1.x". Tesis de Maestría en Informática Aplicada, Fac. Ing. Informática, Inst. Sup. Pol. José Antonio Echeverría, La Habana, Cuba, julio 2010

[19] J. Cácerez Tello, "Curso de Java, Cápsula Formativa. Los interfaces y la herencia múltiple". 2011