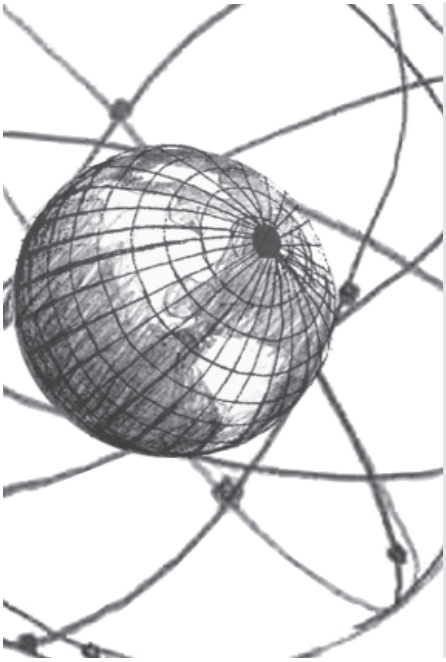
REVISTA Universidad EAFIT
Vol. 43. No. 145. 2007. pp. 45-59

Estudio comparativo de las herramientas MetaCASE bajo consistencia y refinamiento*



Carlos M. Zapata J.
Ph. D. (c) en Ingeniería de Sistemas. Profesor de la Escuela de Sistemas de la Facultad de Minas de la Universidad Nacional de Colombia, Sede Medellín. Integrante del Grupo de Ingeniería de Software de la misma institución. cmzapata@unal.edu.co

Fernando Arango I.
Ph. D. en Informática. Director de Escuela de Sistemas de la Facultad de Minas de la Universidad Nacional de Colombia, Sede Medellín. Integrante del Grupo de Ingeniería de Software de la misma institución. farango@unal.edu.co

Raquel Anaya de Páez
Ph. D. en Informática. Profesora del Departamento de Informática y Sistemas de la Universidad EAFIT y directora del Grupo de Investigación en Ingeniería de Software de la misma Universidad. ranaya@eafit.edu.co.

Recepción: 21 de septiembre de 2006 | Aceptación: 10 de enero de 2007

Resumen

Las herramientas MetaCASE surgieron como una evolución de la tecnología CASE, para superar algunas de las limitaciones identificables en este tipo de tecnologías. Estas herramientas poseen mecanismos que permiten la creación y modificación de paradigmas de metamodelamiento, además de la adición de restricciones que se pueden emplear en chequeos de consistencia o transformaciones de refinamiento. En el artículo que aquí se presenta, se realiza un estudio comparativo de cinco herramientas MetaCASE (AToM^3, DOME, GME, MetaEdit+ y UN-MetaCASE), tomando en cuenta dos puntos de vista: características generales y capacidades para la realización de chequeos de consistencia y transformaciones de refinamiento. El artículo concluye con un análisis comparativo de dichas herramientas.

Palabras Clave

Metamodelamiento
Herramientas MetaCASE
Consistencia
Refinamiento

* Este artículo se realizó en el marco del proyecto de Investigación “Extensiones en herramientas CASE con énfasis en formalismos y reutilización” financiado por COLCIENCIAS, la Universidad Nacional de Colombia y la Universidad EAFIT.

Comparative study of MetaCASE tools under consistency and refinement*

Abstract

MetaCASE tools have emerged as an evolution of CASE technology, for the overcoming of limitations identified in CASE tools. MetaCASE tools have mechanisms for both creation and modification of metamodeling paradigms, in addition to constraints addition—useful in consistency checking or refinement transformations. In this paper, there was a comparative study done on five MetaCASE tools (AToM^3, DOME, GME, MetaEdit+ and UN-MetaCASE), from two points of view: general features and capabilities for consistency checking and refinement transformations. We conclude with some comparative analysis of these tools.

Key words

Metamodeling
MetaCASE Tools
Consistency
Refinement

Introducción

La Ingeniería del Software como disciplina era aún incipiente cuando, a principios de la década de los años ochenta el Dr. John Manley, director del Instituto de Ingeniería del Software de la Universidad de Carnegie Mellon, bautizó como *Computer-Aided Software Engineering* (Ingeniería del Software asistida por computador) a un conjunto de herramientas, procesos y técnicas que se venían gestando desde la década anterior para apoyar diferentes actividades de la Ingeniería del software. Entre ellas se cuentan: modelamiento, construcción, mantenimiento y reingeniería (Burkhard & Jenster, 1989). Desde entonces, las herramientas CASE se han venido posicionando en las actividades de Ingeniería del Software, de forma tal que en la actualidad no es posible pensar en modelos, paradigmas o diagramas del ciclo de vida del software, sin inevitablemente considerar la utilización de una herramienta CASE; en Quintero, Anaya, Marín y Bilbao (2005) se puede apreciar un estudio comparativo de algunas herramientas CASE de la actualidad, donde se incluye AR$_2$CA, que se construye en la actualidad en la Universidad EAFIT.

Una característica cada vez más deseable por parte de las herramientas CASE ha sido la generación automática de código fuente a partir de los diferentes diagramas que modelan la solución informática a un problema. Tal característica, empero, sólo es cumplida parcialmente por las diferentes herramientas CASE disponibles en el mercado, las cuales se han limitado a generar "plantillas" de código fuente que el desarrollador debe complementar para obtener algo que pueda ser similar a un código compilable y ejecutable. Además de esta limitación, las herramientas CASE actuales exhiben otros problemas, como:

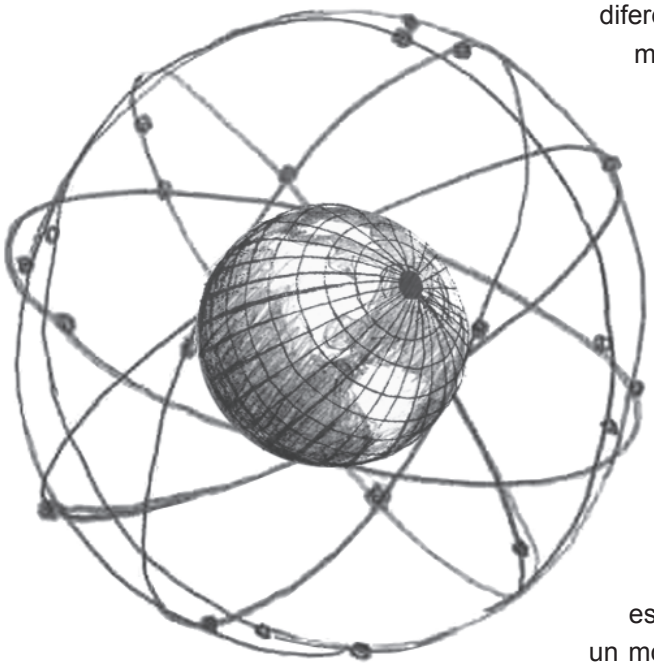
- En general, las herramientas CASE trabajan con paradigmas (conjuntos de conceptos y modelos) definidos para cada herramienta, que no pueden ser modificados por los usuarios finales de dichas herramientas. Por lo general, el *Unified Modeling Language* (UML), definido por el *Object Management Group* (OMG, 2006A), es el estándar con el que más trabajan estas herramientas, pero otros paradigmas de modelamiento que comienzan a tener utilidad en el desarrollo de software, tales como el Diagrama de Procesos, el Diagrama de Objetivos o el Diagrama Causa-Efecto no se suelen incluir en las herramientas CASE

*This article was done under the frame of the research Project "Extension in CASE tools with emphasis on formalisms and reuse" financed by COLCIENCIAS, Universidad Nacional de Colombia, and Universidad EAFIT.

convencionales. Además, las herramientas CASE que trabajan con UML suelen hacerlo con un conjunto reducido de diagramas, o poseen versiones inferiores a la actual 2.0 de ese paradigma.

- Además, por el hecho de no poder modificar los paradigmas cualquier cambio en las especificaciones requiere que se actualice la herramienta CASE completa. Por ejemplo, al cambiar de UML 1.5 a UML 2.0 como estándar se han debido actualizar las diferentes herramientas CASE que se basan en ese formalismo, en lugar de simplemente incorporar las modificaciones en el paradigma de modelamiento correspondiente.

- Otra limitación de las herramientas CASE convencionales se relaciona con la imposibilidad de modificar las restricciones de integridad de modelos, también definidas en las especificaciones de los diferentes paradigmas de modelamiento. Por ejemplo, muchas herramientas CASE excluyen ciertas restricciones como la verificación de duplicidad de nombres o la herencia cíclica del diagrama de clases (OMG, 2006A).



- Finalmente, las herramientas CASE que permiten la generación de código a partir de los modelos construidos en ellas, poseen métodos de traducción muy rígidos, que impiden tomar decisiones de diseño en la traducción.

Las herramientas CASE han evolucionado en otras más flexibles denominadas herramientas MetaCASE; estas últimas no poseen un paradigma de modelamiento definido, sino que es el analista quien debe especificarlo por medio de un modelo de dicho paradigma que se suele denominar "metamodelo". De Lara, Vangheluwe y Fonseca (2003) definen el metamodelamiento como el proceso de construcción del modelo de un determinado formalismo, el cual debe poseer información suficiente como para que una herramienta pueda realizar modelos basados en ese formalismo. La utilidad principal de las herramientas MetaCASE radica en los mecanismos de definición de los paradigmas de modelamiento, que posteriormente permiten que se pueda utilizar la herramienta como se emplean las herramientas CASE convencionales.

Además, en las herramientas MetaCASE diferentes lenguajes posibilitan la adición de restricciones a los paradigmas de modelamiento; es posible lograr que tales mecanismos para el manejo de restricciones se puedan emplear para labores como las siguientes:

- Consistencia. Reglas y relaciones adicionales que deben verificarse entre las partes de un modelo, entre los modelos de una fase y entre los modelos de las diferentes fases del desarrollo.
- Refinamiento. Reglas de transformación que permiten ampliar los modelos en una fase o derivar los de la fase siguiente sin perder la consistencia, por ejemplo las reglas que permiten la generación de código.
- Simplificación. Reglas que permiten derivar modelos reducidos o condensados a partir de uno o varios de los modelos existentes para analizar aspectos particulares del problema modelado.
- Visualización. Formas de visualizar los modelos y sus partes (definición de diagramas).

En este artículo se realiza una revisión de las principales características de cinco herramientas MetaCASE —AToM[3], DOME, GME, MetaEdit+ y UN-MetaCASE—, tomando en consideración sus características generales y las capacidades especiales que poseen en cuanto al chequeo de la consistencia y elaboración de transformaciones de refinamiento.

La estructura del artículo es la siguiente: en la sección 1 se definen la consistencia y el refinamiento, dos de los términos que ayudarán a la evaluación de las capacidades de las diferentes herramientas MetaCASE seleccionadas; en la sección 2 se presentan de manera general esas herramientas, y en la sección 3 se realiza la comparación entre esas herramientas bajo los parámetros seleccionados. Finalmente, se proponen las conclusiones y la visión sobre los trabajos futuros.

1. La consistencia y el refinamiento

Zowghi & Gervasi (2003) realizan un recorrido por las diferentes definiciones que se entregan en la literatura sobre el término "Consistencia". En cuanto a los requisitos que se deben cumplir para una solución informática, la consistencia se define como la carencia de contradicciones entre las especificaciones de dos o más de ellos; dado que los requisitos se originan a partir de las múltiples visiones de los interesados en la pieza de software, se pueden representar en diferentes diagramas y, en este sentido, la consistencia entre esos diagramas es un imperativo para garantizar la buena construcción final de la pieza de software. Existen algunos trabajos donde se definen reglas de consistencia de este tipo, denominadas "intermodelos" por el hecho de que se representa un mismo elemento en diferentes diagramas (Sunetnanta & Finkelstein, 2001).

Además, es necesario que un diagrama particular se adecue a la sintaxis propia del lenguaje en que está escrito, lo que se traduce en que se cumpla con las restricciones internas del diagrama. A este tipo particular de consistencia se le denomina "intramodelo", generalmente ejemplificada con restricciones como la herencia cíclica del diagrama de clases (OMG, 2006 A).

Por lo general, las herramientas CASE no tienen mecanismos que permitan editar reglas de consistencia intra o intermodelos, limitando el chequeo de tales reglas a las que internamente se hayan definido para el paradigma de modelamiento. Uno de los parámetros que se usan en el presente artículo para el estudio comparativo de las herramientas MetaCASE es el mecanismo que se utiliza para la definición de reglas de consistencia intra e intermodelos.

Por otra parte, otro de los estándares definidos por el OMG (2006 B), denominado *Model-Driven Architecture* (MDA o arquitectura orientada por modelos), posibilita la definición de "Refinamiento" como la adición de detalles sobre un modelo para reducir su nivel de Abstracción. En términos del Ciclo de Vida de una pieza de software, que por lo general está constituido por una serie de etapas (definición, análisis, diseño, implementación, pruebas y mantenimiento), el Refinamiento implica el avance hacia etapas más cercanas a la implementación, agregando información necesaria para la realización de ese avance; es decir, se puede pasar desde el análisis hacia el diseño o la implementación mediante el Refinamiento de los

modelos de análisis. Por lo general, la información que se adiciona tiene que ver con las decisiones de diseño que podría utilizar un modelador para representar los modelos de análisis o la información correspondiente a una plataforma de implementación para traducir esos modelos a un lenguaje de programación específico.

Las herramientas CASE no están dotadas con mecanismos que puedan realizar el Refinamiento en los términos descritos y, por ello, se empleará este parámetro también en el estudio comparativo de las herramientas MetaCASE.

En la siguiente sección se realiza una descripción general de las herramientas MetaCASE seleccionadas para el estudio.

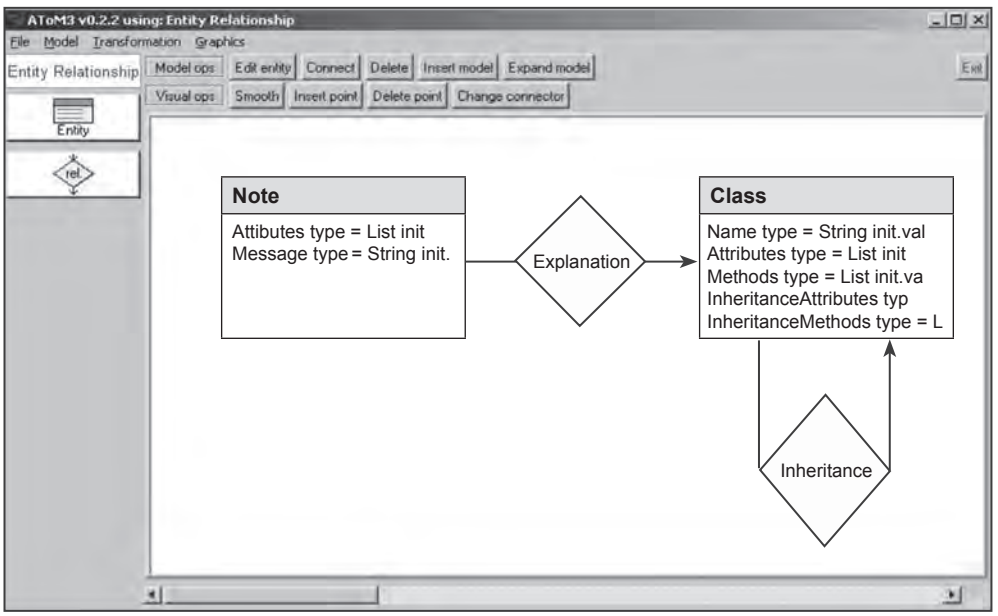
2. Las herramientas MetaCASE

Del ámbito de las herramientas MetaCASE se han seleccionado las siguientes propuestas: AToM^3, DOME, GME, MetaEdit+ y UN-MetaCASE. Las características generales de cada una de las herramientas se listan a continuación.

2.1 AToM^3

AToM3 *(A Tool for Multi-Formalism Modeling and Meta-Modeling)* es una herramienta MetaCASE escrita en el lenguaje de programación Phyton. Esta herramienta posee un mecanismo —basado en el modelo entidad-relación extendido— para la definición de paradigmas de modelamiento en un entorno gráfico, de la misma forma como el usuario elabora los diferentes modelos. Los elementos básicos para definir los paradigmas son las "entidades" que hacen parte del modelo y sus posibles interconexiones o "relaciones". Estos elementos pueden contar adicionalmente con imágenes asociadas a ellos para la construcción de los modelos (De Lara & Vangheluwe, 2002). En la Figura 1 se puede apreciar el metamodelo correspondiente a una versión reducida del diagrama de clases, que incluye dos entidades *("Class" y "Note")* y dos relaciones *("Explanation"* entre *Class* y *Note* e *"Inherintance"* entre entidades del tipo *Class)*.

Figura 1. Formalismo de una versión reducida del diagrama de clases en AToM^3

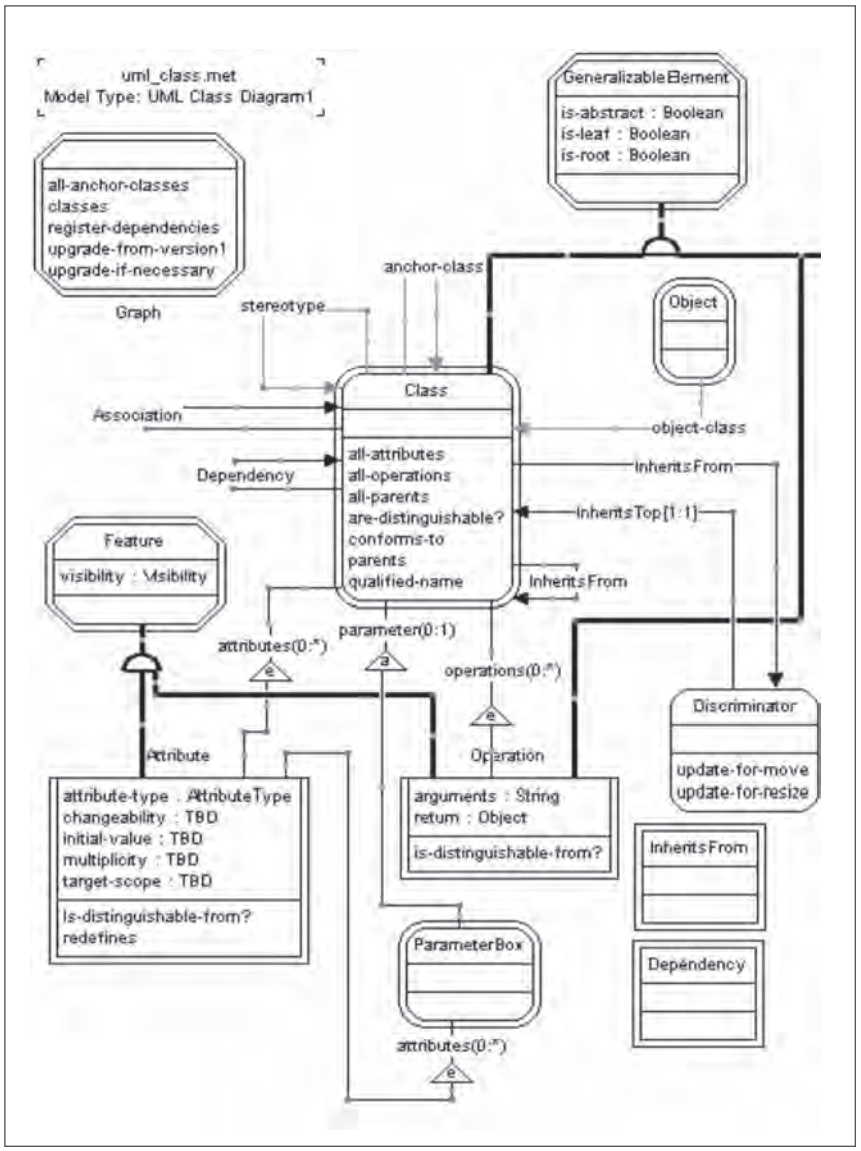


Fuente: Zapata, C. M.; Álvarez, C. y Arango, F. (2005). "Una Propuesta para el Manejo de Restricciones en Modelos de Clases usando AToM3", En: *Revista Ingeniería y Desarrollo*. No. 17.

2.2 DOME

El *Domain Modeling Environment* (DOME, 2006) es una herramienta MetaCASE escrita en Smalltalk que utiliza un lenguaje gráfico—denominado "ProtoDOME"—para la definición de los diferentes elementos del paradigma de modelamiento. Los elementos básicos son diferentes tipos de clases y conexiones, además de otros elementos de control. En la Figura 2 se puede apreciar un metamodelo simplificado del diagrama de Clases en DOME.

Figura 2. Porción del diagrama de Clases simplificado en DOME

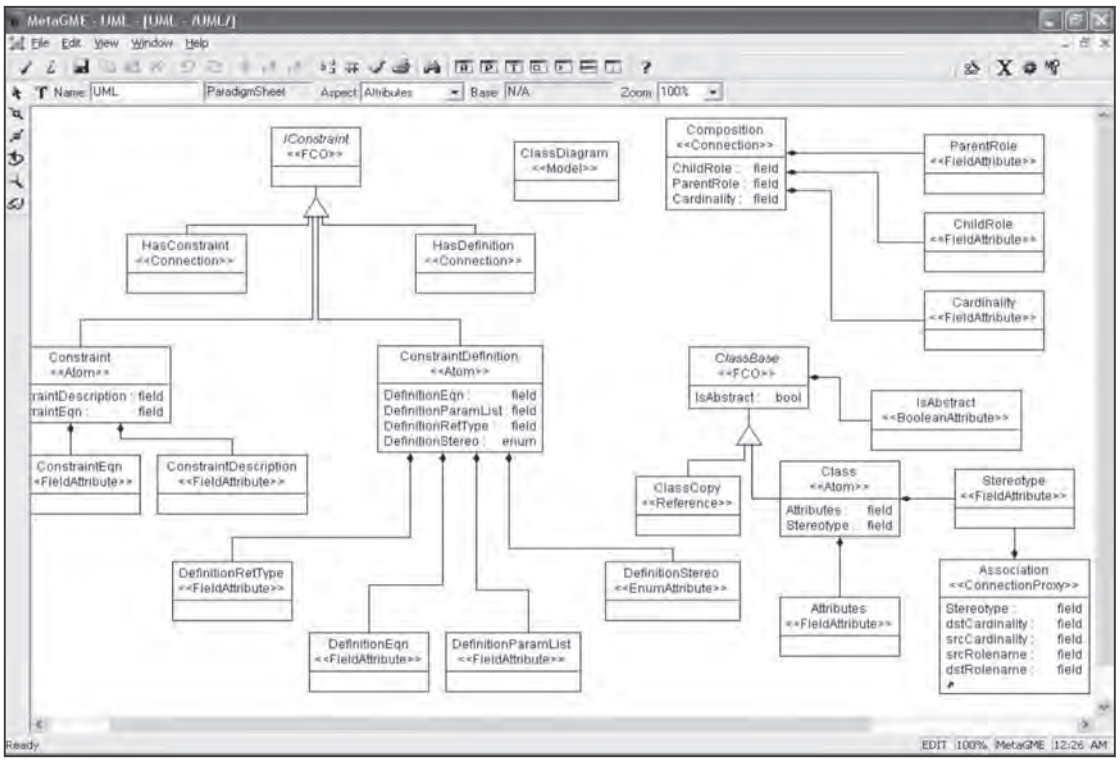


Fuente: Cabarcas, D., Arango, F. y Zapata, C. M. (2006). "Establecimiento y verificación de la consistencia en DOME: Un caso de estudio", En: *Dyna*. No. 148.

2.3 GME

El Generic Modeling Environment (Ledeczi *et al.*, 2001) es una herramienta MetaCASE que utiliza UML como paradigma de modelamiento, incluyendo el manejo de restricciones en el lenguaje OCL *(Object Constraint Language),* como se sugiere en las especificaciones de UML (OMG, 2006A). En la Figura 3 se puede observar el metamodelo del diagrama de clases que viene incluido en las especificaciones de GME; allí se aprecia la misma sintaxis de UML, incluyendo clases, herencias, agregaciones y asociaciones. Se considera también el uso de varios estereotipos, tales como «Model», «Connection» o «FieldAtribute».

Figura 3. Metamodelo del diagrama de Clases incluido en las especificaciones de GME

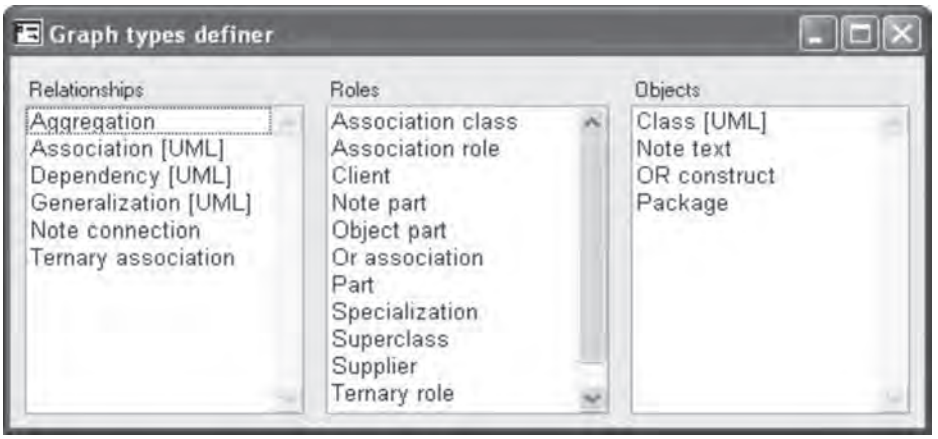


Fuente: Ledeczi *et al.* (2001). "The Generic Modeling Environment". En: *Proceedings of the workshop on intelligent signal processing.* Budapest: IEEE Computer Society.

2.4 MetaEdit+

Es la única herramienta MetaCASE comercial que se seleccionó para este estudio (MetaEdit+, 2006). Como paradigma de modelamiento, esta herramienta utiliza un conjunto de ventanas que incluyen los principales elementos, tales como objetos, relaciones y roles, los cuales se interconectan para formar el metamodelo respectivo. En la Figura 4 se pueden examinar los elementos del metamodelo del diagrama de clases en el formalismo de esta herramienta.

Figura 4. Metamodelo del diagrama de Clases incluido en las especificaciones de MetaEdit+



Fuente: MetaEdit+. (2006). *MetaCase Consulting*, Available: http://www.metacase.com (10 de Septiembre de 2006).

2.5 Un-MetaCASE

Esta herramienta está en proceso de desarrollo en la Escuela de Sistemas de la Universidad Nacional de Colombia a través de varios proyectos de Investigación (Zapata y Arango, 2004). La diferencia fundamental con las demás herramientas MetaCASE descritas radica en el hecho de que en estas herramientas las especificaciones textuales y gráficas de los paradigmas de modelamiento se encuentran ligadas, lo cual quiere decir que la manera de graficar los modelos y la lógica subyacente en ellos no se pueden separar; por el contrario, en el UN-MetaCASE dichas especificaciones se encuentran desligadas, lo que permitiría combinar la especificación lógica de un metamodelo con diferentes expresiones gráficas. En el estado actual del UN-MetaCASE los dos tipos de especificaciones funcionan, de hecho, como programas independientes, y se está trabajando en este momento en las herramientas de comunicación entre ellos. En la Figura 5 se puede apreciar la especificación textual de un metamodelo de clases simplificado, en tanto que en la Figura 6 se muestra parte de la especificación gráfica de ese mismo metamodelo, tal y como se elabora en UN-MetaCASE.

Figura 5. Especificación textual del Metamodelo de un Diagrama de Clases simplificado, elaborada en UN-MetaCASE

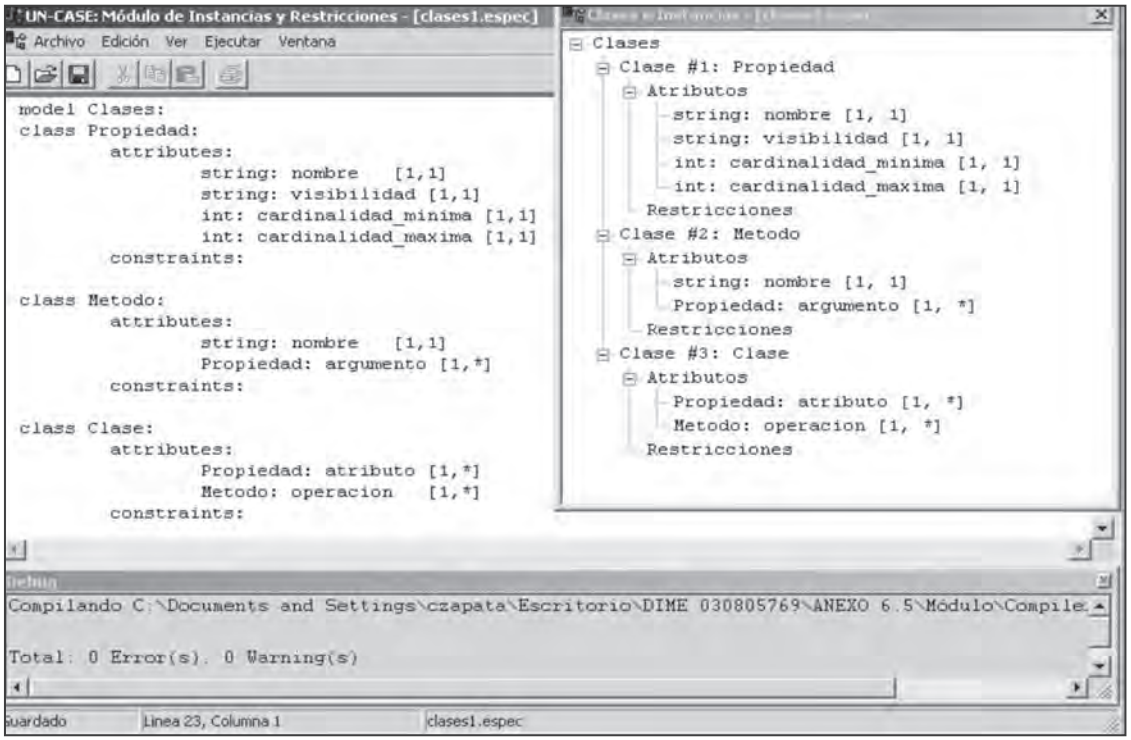
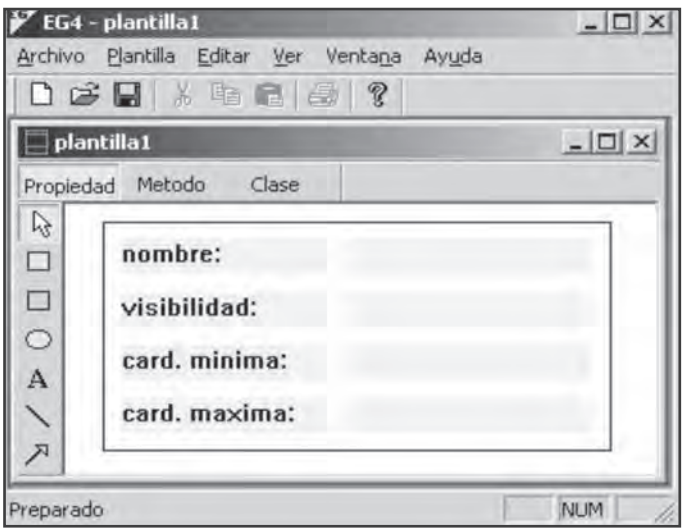


Figura 6. Especificación gráfica de parte del Diagrama de Clases descrito en la figura 5, elaborada en UN-MetaCASE



3. Estudio comparativo de las herramientas MetaCASE seleccionadas

Las herramientas MetaCASE descritas en la sección anterior presentan diferencias en formalismos y mecanismos disponibles, por lo cual se requiere la definición de una serie de criterios que permitan establecer una comparación entre ellas. Dichos criterios se agrupan bajo dos puntos de vista: características generales y mecanismos para la adición de restricciones (útiles en el chequeo de la consistencia y la realización de transformaciones de refinamiento). Además, dentro de cada punto de vista se agruparán los criterios por Lenguajes empleados, Metamodelo y Modelo.

3.1 Características Generales

Bajo este punto de vista, los criterios que se asumen en este estudio son los siguientes:

- Lenguajes Empleados:
  - Lenguaje base: contempla el lenguaje en que fue creada la herramienta.
  - Lenguajes internos: en los cuales es posible realizar acciones al interior de la herramienta.
- Metamodelo:
  - Paradigma de modelamiento: en el cual se pueden elaborar los diferentes metamodelos, que posteriormente serán instanciados en modelos específicos.
  - Visualización gráfica: define la manera en que se puede apreciar gráficamente el metamodelo.
  - Especificación textual: permite establecer si el metamodelo puede ser visualizado de manera textual en algún lenguaje.
  - Independencia de las especificaciones gráficas y textuales: en caso de existir ambas, este criterio establece si los dos tipos de especificaciones necesariamente coexisten o si puede existir independencia entre ellos.
- Modelo:
  - Visualización gráfica. Con este criterio se establece de qué manera se asocian representaciones gráficas a los elementos del modelo.
  - Orientación. Si los modelos que se construyen se refieren a un modelo completo o a diagramas independientes sin conexión.

3.2 Restricciones

Los mecanismos para la verificación de la consistencia y para la realización de transformaciones de refinamiento suelen ser similares. Los criterios seleccionados son:

- Lenguajes empleados:
  - Mecanismos disponibles y lenguajes utilizados: si existe algún tipo de mecanismo dentro de la herramienta MetaCASE que facilite la realización de una u otra tarea (consistencia o refinamiento), al igual que los lenguajes que dan soporte al mecanismo definido.
  - Conocimientos requeridos: este criterio permite establecer cuáles conocimientos son necesarios para especificar las reglas de consistencia o una transformación de refinamiento.
  - Expresividad: establece si la expresión de las restricciones en el lenguaje correspondiente es más larga en relación con su expresión en un lenguaje formal.
- Metamodelo:
  - Transformaciones: si es posible la definición de reglas a nivel de metamodelo que posibiliten la transformación de modelos.

- Momento para la realización del chequeo o transformación: se refiere a la manera de realización del chequeo o transformación, es decir, si se puede realizar en el momento de la edición o si se requiere un botón o menú adicional para realizarlo.

• Modelo:

- Visualización de las especificaciones textuales: si se incluye información adicional a la simple visualización de los elementos.

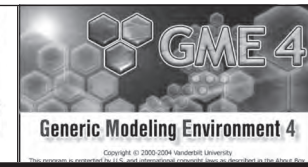
- Adaptación dinámica de representaciones gráficas: se refiere a características como la adaptación automática del tamaño de las cajas al incluir o suprimir texto, la adaptación automática del tamaño del texto al ampliar o reducir elementos y la reubicación automática de figuras (por ejemplo la corrección en el trazado de relaciones cuando se mueven figuras unidas).

- Cambio de representación gráfica: se toma en cuenta si la misma especificación textual puede adaptarse a más de una representación gráfica.

En la Tabla 1 se compendian los valores del estudio comparativo para los dos puntos de vista, tomando en consideración cada uno de los criterios establecidos. Se debe hacer énfasis en que la única herramienta MetaCASE comercial considerada es MetaEdit+ y que UN-MetaCASE es una herramienta aún en desarrollo de la Universidad Nacional de Colombia.

Tabla 1. Estudio comparativo de cinco herramientas MetaCASE bajo Consistencia y Refinamiento

CRITERIO	AToM3	DOME	GME 4 Generic Modeling Environment 4	MetaEdit+	UN MetaCASE
CARACTERÍSTICAS GENERALES: Lenguajes Empleados					
Lenguaje base	Phyton: lenguaje procedural	Smalltalk: lenguaje procedural	Desconocido	Desconocido	C++: lenguaje procedural
Lenguajes internos	Phyton: lenguaje procedural	Alter: versión mejorada del Scheme; es un lenguaje funcional	OCL: lenguaje funcional. C++: lenguaje procedural	OCL: lenguaje funcional; acepta una versión reducida de este lenguaje	UN-LEND: lenguaje funcional y declarativo, propio de la herramienta
CARACTERÍSTICAS GENERALES: Metamodelo					
Paradigma de modelamiento	Entidad-Relación Extendido	Emplea un paradigma propio similar al diagrama de clases pero con muchos elementos adicionales	Emplea un subconjunto del diagrama de clases de UML	Emplea un conjunto jerárquico de ventanas, similar a las herramientas CASE convencionales	Emplea tanto UN-LEND como un conjunto de plantillas gráficas para representar el metamodelo.
Visualización Gráfica	Sí	Sí	Sí	No	Sí
Especificación Textual	Genera archivos para almacenar los metamodelos en Phyton	Genera archivos para almacenar los modelos en Smalltalk	No	No	El archivo en UN-LEND permite almacenar la lógica del metamodelo.
Independencia de las especificaciones gráficas y textuales	No	No	No	No	Sí. Espec. textual en UN-LEND y espec. gráfica en un archivo independiente.
CARACTERÍSTICAS GENERALES: Modelo					
Visualización Gráfica	Posee un Editor Gráfico para elaborar las representaciones de los elementos del modelo	Se debe "describir" en código Alter la imagen asociada con cada uno de los elementos	Posee un Editor Gráfico para elaborar las representaciones de los elementos del modelo	Posee un Editor Gráfico para elaborar las representaciones de los elementos del modelo	Posee un Editor Gráfico para elaborar las plantillas de los elementos del modelo
Orientación	Diagramas independientes	Diagramas independientes	Diagramas independientes	Diagramas independientes	Es posible especificar un conjunto de diagramas en el mismo modelo
RESTRICCIONES: Lenguajes empleados					
Mecanismos disponibles y lenguajes utilizados	Permite la adición de botones con acciones programadas en lenguaje phyton (para consistencia) y posee una "Gramática de Grafos" para la transformación entre modelos (refinamiento)	Permite la creación de botones y opciones de menú con métodos programables en lenguaje Alter (ambos mecanismos se pueden usar en consistencia y refinamiento)	Tiene un editor de restricciones que acepta OCL (para consistencia) y se pueden adicionar botones programables en C++ (para consistencia y refinamiento)	Permite el manejo de algunas restricciones en lenguaje OCL pero muy restringido (puede ser usado para consistencia y refinamiento)	Con UN-LEND se pueden definir modelos y programar algunas de sus restricciones (para consistencia). Aún no tiene mecanismo para el refinamiento

CRITERIO					
RESTRICCIONES: Lenguajes empleados					
Conocimientos requeridos	Lenguajes procedimentales (phyton) y Gramática de Grafos	Lenguajes declarativos y funcionales (Scheme y Alter).	Lenguajes procedimentales (C++) y/o funcionales (OCL)	Lenguajes funcionales (OCL)	Lenguajes declarativos y funcionales (UN-LEND)
Expresividad	Código phyton más largo que expresión formal	Código Alter más largo que expresión formal	Mezcla de C++ y OCL un poco más larga que expresión formal	Código de OCL igual a especificación formal. No es posible expresar todas las restricciones	Código UN-LEND igual a especificación formal
RESTRICCIONES: Metamodelo					
Transformaciones	Sí, aunque parcialmente. Con la Gramática de Grafos se pueden expresar algunas transformaciones, pero se deben complementar con procedimientos en lenguaje phyton	No. Se deben expresar completamente en Alter y Smalltalk	No. Se deben expresar completamente en C++	No tiene contemplado este tipo de transformaciones	No tiene contemplado este tipo de transformaciones
Momento para la realización del chequeo o transformación	Después de edición	Durante o después de edición	Durante o después de edición	Después de edición	Después de edición.
RESTRICCIONES: Modelo					
Visualización de las especificaciones textuales	Sí	Sí	Sí	Sí	Sí
Adaptación dinámica de representaciones gráficas	No cambia el tamaño de los elementos con la adición o supresión de texto, no amplía el tamaño del texto cuando se cambia el tamaño de un elemento y, aunque mantiene unidos los elementos, tiene problemas con la visualización de las relaciones	No amplía el tamaño del texto cuando se cambia el tamaño de un elemento. Las demás sí las realiza bastante bien	Sólo mantiene unidas las relaciones. Las demás no las realiza	Sólo mantiene unidas las relaciones. Las demás no las realiza	Sí las realiza, aunque todavía subsisten algunos problemas
Cambio de representación gráfica	No	No	No	No	Sí

Conclusiones

Todas las herramientas MetaCASE analizadas poseen mecanismos que posibilitan la verificación de la consistencia interna de los diagramas, pero ninguno de ellos posibilita la realización de chequeos de consistencia intermodelos. Una opción viable en todas las herramientas para este tipo de chequeos es la elaboración de un metamodelo conjunto que incluya varios diagramas simultáneamente.

El MetaEdit+ tiene los mecanismos más débiles para el chequeo de la consistencia entre las herramientas metaCASE analizadas, puesto que sólo permite chequear expresiones muy simples en OCL ante la creación de ciertos elementos en el modelo.

La separación entre las especificaciones gráfica y textual, sólo presente en el UN-MetaCASE para las herramientas analizadas, puede ser una característica deseable en el análisis de un modelo, puesto que permite la reutilización de una especificación textual con diferentes representaciones gráficas y viceversa.

A nivel de refinamiento, el mejor mecanismo de las herramientas analizadas es la Gramática de Grafos, correspondiente al AToM^3. Este mecanismo permite la conversión entre diagramas con diferente metamodelo, lo que puede llevar paulatinamente a la generación de código en diferentes lenguajes.

Entre las herramientas analizadas, el mejor mecanismo de visualización de modelos corresponde al DOME, puesto que permite, por ejemplo, corregir la ortogonalidad entre líneas y adaptar automáticamente el tamaño de algunos elementos cuando se añade información en su interior, como suele ocurrir en algunas de las herramientas CASE convencionales. Paradójicamente, entre todas las herramientas analizadas, el DOME posee la forma más compleja de definición de la representación gráfica de los elementos, puesto que no cuenta con un editor gráfico como las demás herramientas, sino que se debe recurrir a la definición de métodos en lenguaje Alter para definir cada elemento gráfico.

A nivel de estándares de modelamiento reconocidos, sólo GME y MetaEdit+ manejan algunos elementos del lenguaje OCL; las demás herramientas emplean lenguajes que no se utilizan por lo general en la descripción de modelos. A nivel gráfico, sólo GME hace uso del diagrama de Clases de UML como paradigma de metamodelamiento.

Trabajo futuro

Si bien el estudio comparativo se realizó sobre cuatro herramientas MetaCASE disponibles en la Web y de fácil obtención (AToM^3, DOME, GME y MetaEdit+) y una herramienta de tipo académico actualmente en construcción (UN-MetaCASE), es importante ampliar la base de herramientas MetaCASE, ensayando restricciones de consistencia y transformaciones de refinamiento en otras herramientas, ya sean comerciales o académicas.

El proyecto de construcción de UN-MetaCASE se ha nutrido de la experiencia de trabajo con otras herramientas MetaCASE y de allí derivan algunos de los elementos para desarrollar en futuros proyectos de investigación como:

- Un mecanismo para la transformación entre modelos, similar a la Gramática de Grafos de AToM^3.
- Un módulo para la obtención de especificaciones textuales a partir de especificaciones gráficas.
- Mejoramiento de las capacidades gráficas del editor, mediante la adición de mecanismos como la corrección automática de la ortogonalidad de las líneas que emplea el DOME.
- Incorporación de estándares más aceptados de modelamiento, como OCL y UML, de manera similar al uso que de ellos hace el GME.
- Definición de un mecanismo para el chequeo de la consistencia intermodelos a partir de diferentes metamodelos.

Bibliografía

Burkhard, D. & P. V. Jenster. (1989). "Applications of Computer-Aided Software Engineering Tools: Survey of Current and Prospective Users". En: *Data Base*. Vol. 20, No. 3. pp. 28-37.

Cabarcas, D., Arango, F. y Zapata, C. M. (2006). "Establecimiento y verificación de la consistencia en DOME: Un caso de estudio". En: *Dyna*. No. 148. pp. 17-28.

De Lara, J., & H. Vangheluwe. (2002). "AToM3: A tool for Multi-Formalism and Meta-Modeling". En: *Proceedings of the Fifth International Conference on Fundamental Approaches to Software Engineering*. Grenoble: Ralf-Detlef Kutsche, Herbert Weber (Eds.). pp. 174-188.

De Lara, J.; Vangheluwe, H. & Alfonseca, M. (2003). "Using Meta-Modelling and Graph Grammars to create Modelling Environments". En: *Electronic Notes in Theoretical Computer Science*. Vol. 72. No. 3.

DOME. (2006). *What is Dome*. [From: http://www.htc.honeywell.com/dome/description.htm] (10 de Septiembre de 2006).

Ledeczi, A. *et al.* (2001). "The Generic Modeling Environment". En: *Proceedings of the Workshop on Intelligent Signal Processing*. Budapest: IEEE Computer Society.

MetaEdit+. (2006). *MetaCase Consulting*, [From: http://www.metacase.com] (10 de Septiembre de 2006).

OMG. (2006A). *OMG Unified Modeling Language Specification*. *Object Management Group*. [From: http://www.omg.org/UML/] (10 de Septiembre de 2006).

OMG. (2006B). *OMG Model-Driven Architecture*. *Object Management Group*. [From: http://www.omg.org/MDA/] (10 de Septiembre de 2006).

Quintero, J. B.; Anaya, R., Marín, J. C. y Bilbao, A. (2005). "Un Estudio Comparativo de Herramientas para el Modelado con UML". En: *Revista Universidad Eafit*. Vol. 41. No. 137. pp. 60-76.

Sunetnanta, T. & A. Finkelstein. (2001). "Automated Consistency Checking for Multiperspective Software Specifications, *Workshop on Advanced Separation of Concerns*, The 23rd International Conference on Software Engineering (ICSE2001), Toronto: IEEE Computer Society.

Zapata, C. M. y F. Arango. (2004). "UN-LEND: Un lenguaje para la especificación de modelos de UN-MetaCASE". En: *Avances en Sistemas e Informática*. Vol. 1. No. 2. pp. 7-19.

Zapata, C. M.; Álvarez, C. y Arango, F. (2005). "Una Propuesta para el Manejo de Restricciones en Modelos de Clases usando AToM3". En: *Revista Ingeniería y Desarrollo*. No. 17. pp. 130-147.

Zowghi, D. & V. Gervasi. (2003). "On the Interplay between Consistency, Completeness, and Correctness in Requirements Evolution". En: *Information and Software Technology.* Vol. 45. No. 14. pp. 993-1009.