Herramientas de soporte al proceso de desarrollo dirigido por modelos y su implementación con DSL Tools

L. Cuaderno, E. Di Lorenzo, A. Gaig, D. García, R. Giandini
L. Nahuel, L. Ocaranza, M. Pinasco, C. Pons, F. Salvatierra

Universidad Tecnológica Nacional, Facultad Regional La Plata
La Plata, Buenos Aires, Argentina - proyectopampa@frlp.utn.edu.ar

Abstract
El uso de modelos para construir distintos tipos de sistemas de software es actualmente una de las claves para desarrollar nuevas tecnologías. La idea fundamental de este "desarrollo dirigido por modelos" (MDD) es sustituir, como artefacto principal en el proceso de desarrollo de software, al código de lenguajes de programación por modelos. De este modo, tales modelos son considerados como entidades primordiales, permitiendo nuevas posibilidades de crear, analizar y manipular sistemas a través de diversos lenguajes y herramientas. Dentro de este contexto que involucra lenguajes gráficos de modelado, la especificación de modelos precisos, herramientas de transformación de modelos y lenguajes para especificar modelos formales, entre otros, yace la necesidad de crear PAMPA: una herramienta de entorno visual - desarrollada sobre plataforma .NET que permite construir modelos a partir de diagramas UML, y provee soporte formal con expresiones, basados en un metamodelo de especificación. Mostraremos paso a paso las incumbencias, funcionalidades y ámbito de aplicación de la herramienta, la cual se orienta esencialmente, con fines educativos, a fomentar el uso del nuevo paradigma MDD. También expondremos como es posible implementar PAMPA haciendo uso de los beneficios de las DSL Tools de Microsoft.

Palabras Clave
Desarrollo dirigido por modelos (MDD), Herramientas para definir Lenguajes de Dominio Específico (DSL Tools), Lenguaje unificado de modelado (UML), Lenguaje de especificación formal para objetos (OCL).

1 Introducción

El uso de modelos para diseñar y construir distintos tipos de sistemas es actualmente una de las claves para desarrollar nuevas tecnologías en áreas de la industria del Software.

Considerando que un proceso es un conjunto de pasos parcialmente ordenados conducidos hacia una meta, la Ingeniería de Software se propone construir un producto de software o mejorar uno existente, mientras que la Ingeniería de Procesos tiene como fin desarrollar o mejorar un proceso. En ambos casos se hace uso del modelado para cumplir el objetivo correspondiente.

Considerando a un modelo como una "una interpretación simplificada de la realidad" (Eric Evans), es posible destacar el hecho de que se trata de una "interpretación", y no de una correspondencia entre el modelo y la porción de realidad que se intenta modelar. Esto es de fundamental importancia - específicamente en el modelado de sistemas de software - para poder definir modelos precisos y acordes a la esencia del problema, ya que el modelo representa un lazo común de comunicación entre los desarrolladores, diseñadores, expertos y usuarios en un determinado dominio.

Un factor de fuerte influencia en la fidelidad de los modelos es el grado de formalismo del lenguaje de modelado utilizado. Combinando las ventajas de las notaciones gráficas intuitivas y los formalismos matemáticamente exactos, el usuario puede interactuar con un lenguaje gráfico, confiando en la base formal que el esquema matemático subyacente le proporciona. De este modo se introduce precisión al proceso de modelado en un ámbito de desarrollo de software, conservando la usabilidad y aceptación por parte de los desarrolladores. Combinar estas ventajas conduce a una mejor comprensión de conceptos, permitiendo un uso eficaz de las tecnologías involucradas.

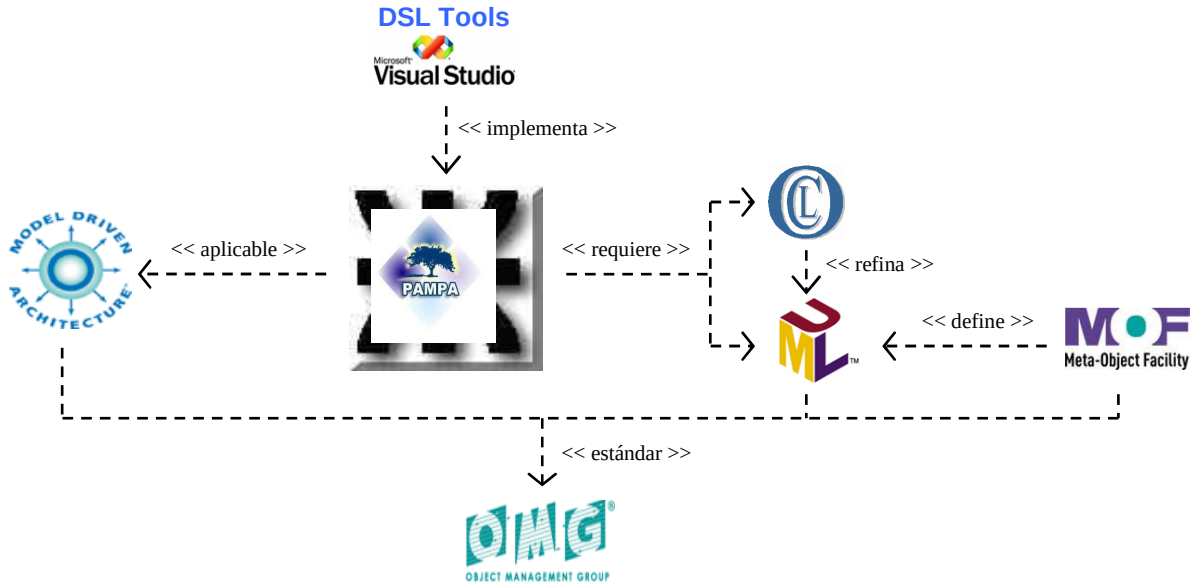

Figura 1: Relación entre las bases conceptuales de PAMPA

Apoyándonos en: el estudio y análisis del nuevo paradigma MDD (Model Driven Development) [1], que alienta hacer de los modelos valores conductores primarios en todos los aspectos del desarrollo de software; un lenguaje gráfico de modelado como UML (Unified Modeling Language) [2] y un lenguaje formal para especificar restricciones fuertemente aceptado como OCL (Object Constraint Language) [3]; presentaremos en este artículo las características y funcionalidades de la herramienta educativa **PAMPA** (Precise Assistance for the Modeling Process Activities) [4], que apoya la filosofía MDD, aplicando y combinando notaciones gráficas de UML con fundamentos formales a través de OCL.

La construcción de esta herramienta se encuentra fortalecida en el estudio y combinación de estándares de la industria del software. La Figura 1 intenta reflejar la integración de estos estándares, como así también el rol que cumplirán las DSL Tools de Microsoft [15] en la herramienta. El Proyecto PAMPA tiene como finalidad abordar metas específicas, tales como: análisis conceptual de las técnicas de modelado de software; estudio y formalización de lenguajes para crear modelos; definición y formalización de lenguajes para describir transformaciones, como así también implementar herramientas de software automáticas.

El resto del artículo se organiza de la siguiente manera: en la sección 2 se presenta la metodología MDD describiendo los componentes del estándar MDA (Model Driven Architecture) [5], en la sección 3 se exponen las funcionalidades de PAMPA, mientras que en la sección 4 se expone como será la implementación de PAMPA haciendo uso de la novedosa tecnología DSL Tools de Microsoft. En la sección 5 se destacan los aspectos educativos de la herramienta propuesta y finalmente en la sección 6 se resaltan las líneas de trabajo futuro y conclusiones finales.

2 Desarrollo dirigido por modelos

La visión del “Desarrollo de software dirigido por Modelos” (MDD) se ha convertido actualmente en un importante paradigma de la Ingeniería de Software con el fin de manipular amplias complejidades y requerimientos de sistemas con gran cantidad de software.

La idea fundamental de MDD es sustituir al código de lenguajes de programación específicos por modelos. De este modo y en el contexto de este paradigma, los modelos son considerados como entidades de

primera clase, permitiendo nuevas posibilidades de crear, analizar y manipular sistemas a través de diversos tipos de herramientas y de lenguajes.
Cada modelo trata generalmente un aspecto, y las transformaciones entre modelos proporcionan un vínculo que permite la implementación automatizada de un sistema a partir de sus correspondientes modelos, generando también modelos, por lo tanto constituyen una parte integral de esta propuesta basada en modelos. Estas transformaciones requieren soporte especializado en varios aspectos para que: modeladores de sistemas, desarrolladores de transformaciones y desarrolladores de herramientas puedan aplicarla en su máximo potencial. Este es el eje central de la iniciativa MDA del OMG (Object Management Group, organización que promueve estándares para la industria del software) [6], y es actualmente un problema abierto que continúa siendo parcialmente refinado por nuevas propuestas.
MDA propone definir un conjunto de normas no propietarias que especifican las tecnologías de transformación con las cuales llevar a cabo el desarrollo conducido por modelos mediante transformaciones automatizadas. No todas estas tecnologías están involucradas directamente en las transformaciones incluidas en MDA.
No todas estas tecnologías están involucradas directamente en las transformaciones incluidas en MDA. Esta arquitectura no está necesariamente basada en UML, pero, como un tipo especializado de MDD, necesariamente involucra el uso de modelos en el desarrollo, la cual requiere el uso de al menos un lenguaje de modelado. Cualquier lenguaje de modelado empleado en esta arquitectura se debe describir en términos del lenguaje MOF (Meta-Object Facility) [7] - otro estándar de OMG para dar soporte a la visión MDA - para asegurar que se pueda entender el metamodelo de una manera estándar, lo cual es requisito para poder realizar cualquier transformación automática.
La relación entre los grandes componentes del framework MDA puede observarse en la Figura 2.
El framework MDA tiene dos ejes principales:

1º) Hace énfasis en la separación entre la especificación de las funcionalidades del sistema y la implementación de las mismas utilizando plataformas tecnológicas concretas. Para ello, MDA identifica tres tipos de modelos, mostrados en la Figura 2, que representan los artefactos centrales en el ciclo de vida de desarrollo en un ambiente de software

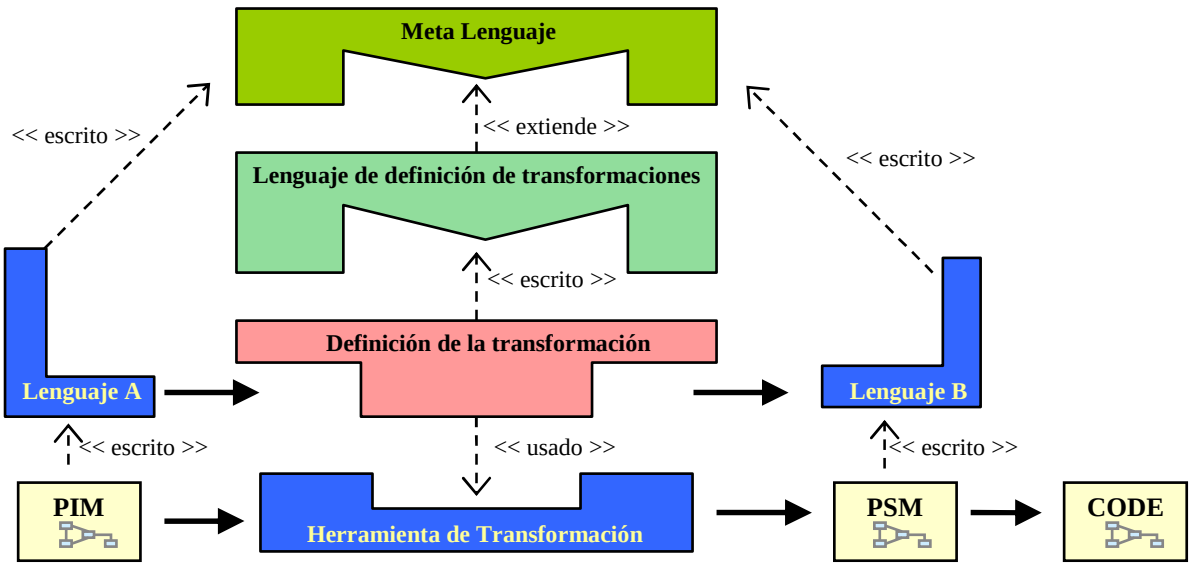

Figura 2: Integración de los componentes del framework MDA

dirigido por modelos:

- ***Modelo Independiente de la Plataforma (PIM,*** *Platform Independent Model):* modelos con alto nivel de abstracción e independientes de cualquier tecnología de implementación.
- ***Modelo Específico de la Plataforma (PSM****, Platform Specific Model):* modelos que especifican el sistema en términos de construcciones de implementación (código) dependiendo de una tecnología específica.
- ***Código (CODE):*** código fuente del modelo escrito en un lenguaje de programación determinado.

2º) Los modelos son considerados conductores primarios en todos los aspectos del desarrollo de software. Un PIM puede ser transformado en uno o más PSMs, es decir que para cada plataforma tecnológica específica se genera un PSM preciso. La transformación entre modelos constituye el motor de MDA y de esta manera los modelos pasan de ser entidades meramente contemplativas a ser entidades productivas.

Una transformación se define a partir de una colección de reglas de transformación, las cuales son especificaciones no ambiguas de la forma en que un modelo puede ser usado para crear otro modelo. De acuerdo a esta visión, las transformaciones llegan a su máxima potencialidad al ser realizadas por herramientas automáticas que pueden mantener información sobre el proceso de generación y ser vistas como objetos complejos que contienen enlaces tanto con los modelos de entrada como así también con los modelos de salida.

Un desarrollador o modelador se focaliza así en la construcción de un PIM, describiendo el software en un alto nivel de abstracción. Luego elige una o más herramientas que le permitan ejecutar una transformación automática del PIM desarrollado, de acuerdo a ciertas definiciones de transformación. Esto resulta en un PSM, el cual puede ser luego transformado en código fuente.

Todo esto significa un aumento de productividad en dos formas:

1º → Los desarrolladores de PIMs tienen menos trabajo debido a que omiten detalles específicos de las plataformas.

2º → Los diseñadores pueden focalizarse en los PIMs antes que en el código, prestando mayor atención a resolver problemas de negocio.

Tal ganancia de productividad prometida por la visión MDA sólo puede ser alcanzada con el uso de herramientas que automaticen totalmente la transformación PIM – PSM, lo cual exige que mucha de la información acerca de la aplicación debe ser incorporada en el PIM y/o en la herramienta de generación. En consecuencia, dado que el modelo de alto nivel se encuentra directamente relacionado con el código a generar, la demanda de completitud, corrección y consistencia de estos PIMs es mucho mayor que en el desarrollo tradicional de software.

MDA involucra el uso de modelos en el desarrollo, por lo cual se debe usar al menos un lenguaje de modelado. Para poder ser utilizado bajo esta arquitectura estándar, el lenguaje de modelado debe ser descripto en términos del lenguaje MOF, permitiendo así que los metadatos sean entendidos de manera estándar. Esto es precondición para poder realizar las transformaciones automáticas.

UML es el lenguaje estándar especificado por OMG para visualizar, especificar, construir y documentar los artefactos de un sistema (sirve además para el modelado del negocio y sistemas no software). Este se ha convertido actualmente en la notación de facto para el modelado de sistemas con gran cantidad de software. Tiene como ventajas primordiales: poder usarse en las diferentes etapas del ciclo de vida del desarrollo de sistemas, ser independiente del proceso o metodología de desarrollo y del lenguaje de implementación. Al ser un lenguaje fuertemente visual, permite crear y

modificar modelos expresivos mediante la combinación de sus 13 tipos de diagramas que suministra en su versión 2.0. Es posible además extender la funcionalidad de la notación gráfica mediante estereotipos y proveer una base formal para los diagramas a través de OCL.
UML es actualmente la mejor alternativa para modelar PIMs y PSMs (usando para este último caso, alguno de los variados PROFILES que especializan un modelo UML para representar cierta tecnología). Surge entonces una fuerte necesidad de aprendizaje y dominio profundo de este lenguaje de modelado, a fin de insertarse y ser útil en la visión MDA.
Actualmente algunos de estos componentes están bien fundamentados y se están empezando a aplicar con éxito. Otros sin embargo, están todavía en proceso de definición.

2.1 Necesidad de métodos formales en ambientes MDA

La transformación de PIMs no ejecutables a PSMs ejecutables generalmente implica pérdida de algunas de las fortalezas del modelado. Mientras que los lenguajes de modelado (como UML) tratan, por ejemplo, a las relaciones como características muy importantes en el modelo, tal conocimiento se pierde en los lenguajes utilizados para la definición del PSM correspondiente (como Java). Además, como la semántica formal de estos lenguajes de modelado aún presenta debilidades en su especificación estándar, el mapeo de PIM a PSM debe ser definido para cada implementación particular.
Surge entonces la necesidad de lograr modelos enriquecidos a fin de que las transformaciones automáticas planteadas por MDA puedan ser soportadas y logradas en su totalidad. UML encuentra el aliado perfecto para esto en OCL, un lenguaje formal para escribir reglas y restricciones sobre objetos. Toda expresión escrita en OCL se basa en tipos primitivos y tipos definidos en los diagramas UML (como clases, interfaces, etc.). Por lo tanto, su uso incluye la utilización de algunos aspectos de UML y le agrega a sus diagramas información que no puede ser expresada de otra manera. Si bien en las primeras versiones - UML 1.1 [8] - esta información estaba limitada a restricciones sobre valores del modelo, en la versión 2.0 la premisa es que la información adicional debe ser mucho más que meras restricciones. La definición de consultas, referencia de valores, establecimiento de condiciones o reglas de negocio, debe estar acompañada de la escritura de expresiones OCL. Este es entonces, el lenguaje estándar en el cual dichas expresiones pueden ser escritas de manera clara y no ambigua.
La combinación de UML con OCL en la construcción de PIMs mejora la calidad de éstos, es decir, se obtienen PIMs consistentes, con información suficiente y precisa (esto garantiza que las herramientas de transformación puedan entenderlos). El fuerte aspecto estructural de UML puede ser dotado de mayor consistencia y completitud. Las expresiones OCL pueden utilizarse para definir reglas de buenas formación a nivel de modelo y/o metamodelo. Una definición de transformación podría ser fácilmente escrita por dos expresiones OCL: una para identificar el elemento en el modelo de entrada que será transformado, y otra para identificar el elemento en el modelo de salida que necesita ser generado.
Si bien la dinámica del sistema aún no puede ser totalmente especificada con la combinación UML-OCL (aunque sí en parte, mediante el uso de “pre y post condiciones" en las operaciones), esta última permite generación de PSMs y código ejecutable mucho más precisos que utilizando solamente UML.

3 PAMPA: un aporte valioso a MDA

La comprensión formal de un lenguaje facilita un uso apropiado de las herramientas CASE de UML. Varias herramientas CASE se basan en la definición formal de la sintaxis y semántica de UML. Suele ser más fácil que los

alumnos o usuarios entiendan la herramienta si conocen UML formalmente. Sin este conocimiento, los alumnos o usuarios pueden utilizar las herramientas pero no perciben el "por qué" detrás del comportamiento. Por ello hemos propuesto construir PAMPA, una herramienta CASE para el desarrollo de software basado en modelos que usa notaciones gráficas con fundamentos formales.

El objetivo de este proyecto es que sea una herramienta educativa, poniendo énfasis en la comprensión y aprendizaje de los componentes fundamentales de MDA, facilitando así la incorporación de aprendizaje de lenguajes como UML y OCL.

PAMPA pretende dar soporte al aprendizaje de las tecnologías MDA y proveer el medio para construir modelos de transformación claros, precisos y estándares, dentro del marco de las buenas prácticas que promueve MDA. Con el uso de la herramienta: a) se logra una comprensión más profunda de los conceptos expresados mediante la notación, permitiendo un uso más maduro de la tecnología; b) se reducen las malas interpretaciones del modelo y c) aumenta el poder expresivo del lenguaje de modelado mediante el uso de reglas OCL, que permiten expresar y verificar propiedades específicas del modelo.

3.1 Funcionalidades provistas por PAMPA

La herramienta provee soporte a los siguientes aspectos:

- *Edición visual de Modelos:* Permite la creación gráfica de diagramas UML, que son evaluados, al mismo tiempo, mediante reglas expresadas en OCL. La formalización de los diagramas se obtiene a través del metamodelo subyacente.
- *Persistencia de Modelos*: La herramienta es independiente del mecanismo de persistencia, proveyendo acceso uniforme a varios repositorios, incluyendo MDR [9]. Los Modelos se almacenan y recuperan de un repositorio orientado a objetos.
- *Evaluación OCL*: El evaluador toma expresiones OCL y las evalúa sobre un modelo dado. El proceso de evaluación y chequeo permite visualizar errores de validación, indicando los elementos del modelo que no cumplen las reglas de buena formación que se encuentran definidas en los archivos de OCL.
- *Traducción a Z:* A partir de un modelo UML enriquecido con OCL se genera una especificación en el lenguaje formal Z [10]. La notación de este lenguaje está basada en la teoría de conjuntos y lógica de predicado de primer orden. Esta formalización permite aplicar técnicas clásicas de verificación y pruebas de teoremas sobre los modelos de usuarios. PAMPA genera automáticamente la especificación Z escrita en Z-Latex [11] para facilitar el intercambio de ésta entre diversas herramientas.

3.2 Componentes de PAMPA

Las funcionalidades antes mencionadas se implementan concretamente en base a los siguientes componentes, que se muestran en la Figura 3 y se describen a continuación:

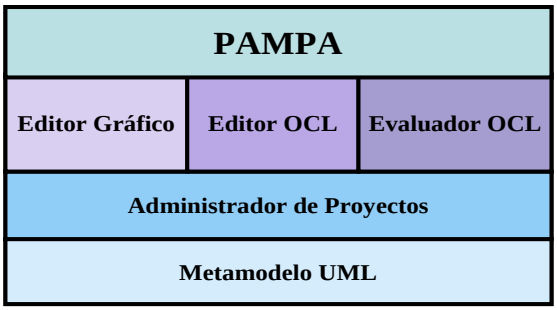

Figura 3: Componentes de PAMPA

→ ***Metamodelo UML:*** La arquitectura de MOF está basada en una estructura de cuatro capas: objetos de usuarios, modelo, metamodelo y meta-metamodelo. El metamodelo UML es un modelo lógico que define un lenguaje para expresar otros modelos, pero no un modelo físico o de implementación. Esto tiene como ventaja hacer énfasis en las semánticas declarativas

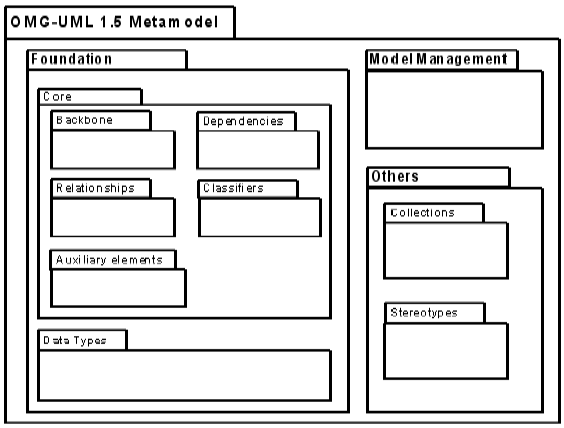

Figura 4: Organización del metamodelo en paquetes

y no en los detalles de implementación. Aquellas implementaciones que usen el metamodelo lógico deben ajustarse a su semántica y también deben ser capaces de importar y exportar modelos en forma parcial y total. La desventaja, es que se carece de semánticas imperativas requeridas para implementaciones eficientes y precisas. Por esta razón, el metamodelo usado para la herramienta PAMPA enriqueció al metamodelo original -UML v1.5- para posibilitar una implementación física basada en una jerarquía de clases en C# [12] y mecanismos para manipulación de tipos de datos, colecciones y elementos gráficos. En la Figura 4 se observan los paquetes principales del metamodelo UML en los que está basada la implementación.

→ ***Administrador de Proyectos****:* Es la capa intermedia de la herramienta, que sirve de nexo entre la implementación del metamodelo y las capas de visualización, modelado y evaluación. En la misma se incluye: la administración y serialización de los proyectos que se realizan con la herramienta, consultas complejas a los objetos del repositorio, control de cambios y recuperación de objetos del repositorio basada en los tipos especificados del metamodelo. El mismo está implementado con una base de datos de código abierto orientada a objetos (DB4O [13]).

→ ***Editor gráfico****:* Este editor permite la

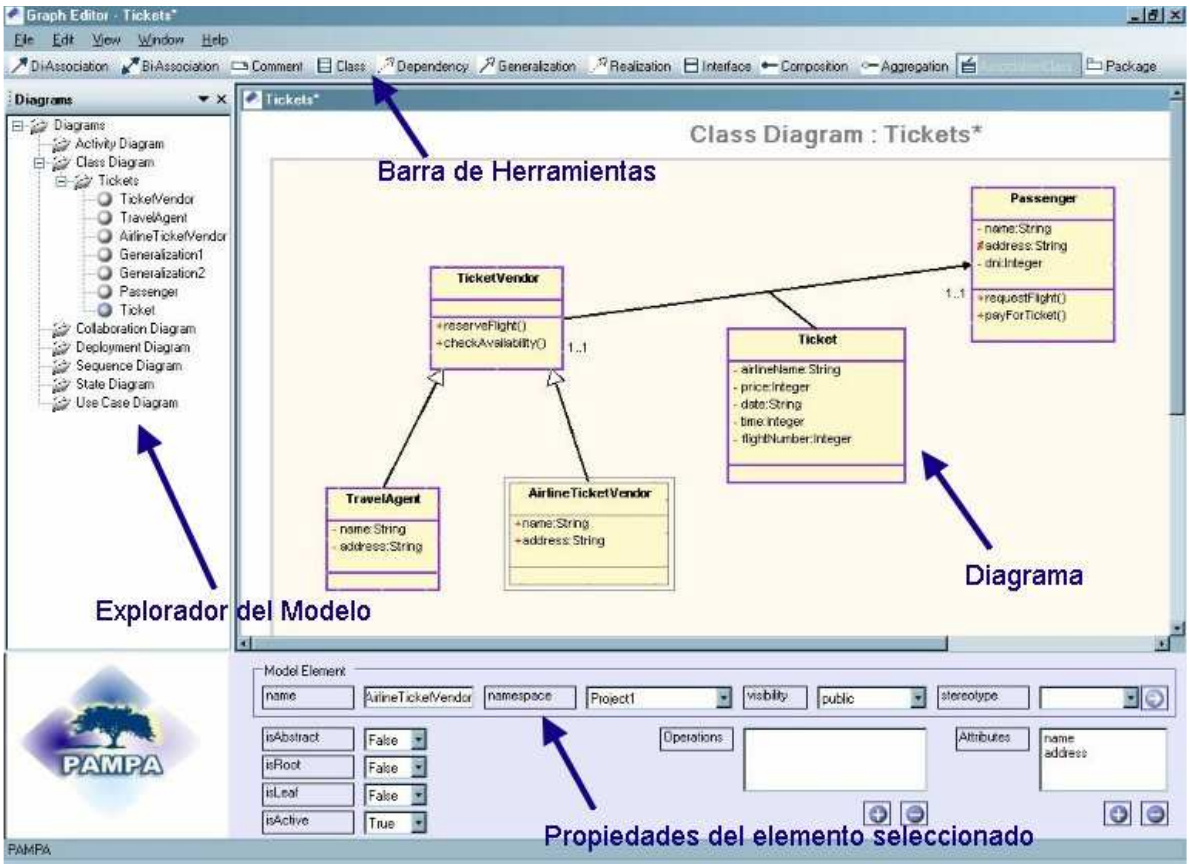

Figura 5: Editor Gráfico

Figura 6: Barra de Herramientas

creación de modelos a través de diagramas UML. Al igual que otras herramientas ampliamente difundidas, se basa en el manejo intuitivo de elementos gráficos (estilo *drag & drop*), contribuyendo así al fácil aprendizaje de las elaboraciones de modelos mediante notación UML. Está integrado por cuatro secciones principales, como se observa en la Figura 5 y se describe a continuación:

- El ***Explorador de Modelos***, que nos permite navegar el proyecto mediante dos perspectivas: Paquetes y Diagramas.
- La ***Barra de Herramientas*** - ampliada en la Figura 6 -, es el lugar donde se encuentran todos los elementos que se pueden agregar al modelo, tales como: clases, interfaces, asociaciones, paquetes, etc.
- El ***área central***, que constituye el espacio de trabajo donde se realizan los diagramas.
- El ***área de propiedades*** - ampliada en la Figura 7 -, donde se visualizan y modifican los atributos de aquellos elementos que formen parte de nuestro modelo. Para acceder a estas propiedades basta con seleccionar un elemento del diagrama correspondiente (o a través del Explorador de Modelos). La forma en que se muestran es a través de campos editables y combos desplegables, lo cual posibilita un acceso rápido e intuitivo.

→ ***Editor de Fórmulas y Evaluador OCL***: Permite editar y evaluar fórmulas OCL que son aplicadas al diagrama UML que se está modelando. Dichas fórmulas se pueden aplicar en dos niveles: metamodelo y modelo. A nivel metamodelo, la especificación de UML provee reglas de buena formación que deben cumplirse en cualquier modelo UML. A diferencia de otras herramientas de edición UML donde el chequeo de correcciones sintácticas de modelos UML es automático, PAMPA provee un chequeo aún mas detallado a fin de que el usuario visualice claramente no solo los errores sintácticos del modelo, sino también su justificación formal (a través de una expresión OCL) incorporando así el aspecto pedagógico que se pretende dar a la herramienta. Este editor se divide en tres secciones principales, como se observa en la Figura 8 y se describen a continuación:

En el ***Espacio de Trabajo*** se pueden escribir las fórmulas OCL y verificar su sintaxis.

En la ***Vista de Errores*** el editor reporta los errores producidos al evaluar las fórmulas OCL incorrectas sintácticamente mediante markers (indicadores gráficos).

En el ***Explorador de Archivos*** se visualiza la estructura de los archivos de extensión OCL, que contienen las

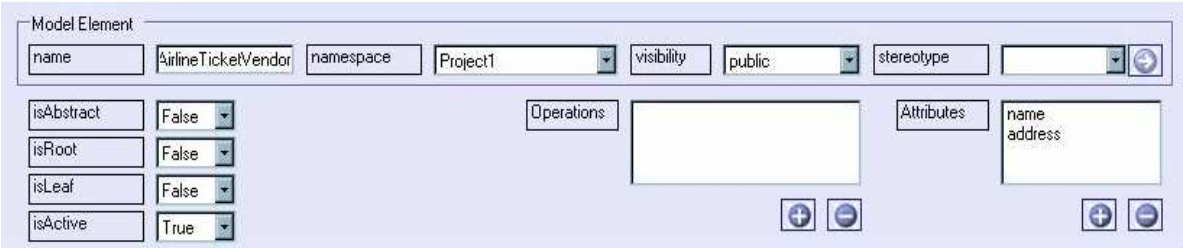

Figura 7: Propiedades del elemento seleccionado

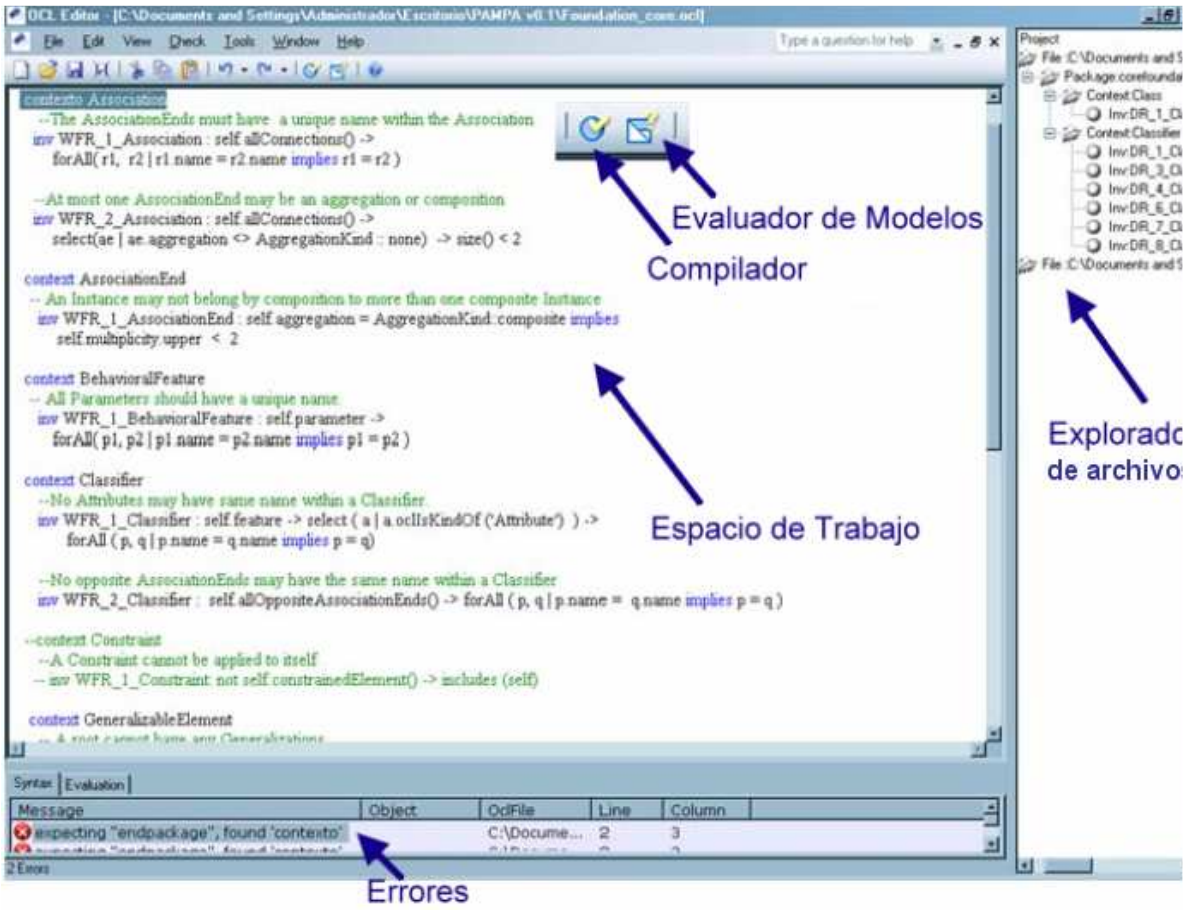

Figura 8: Editor de Fórmulas y Evaluador OCL

expresiones aplicadas por el evaluador para el proyecto que se está validando. A nivel de metamodelo el evaluador se basa en dos archivos: Design.ocl, el cual contiene las reglas del diseño, y FoundationCore.ocl, que contiene las reglas de la buena formación (well-formedness) de UML. A nivel de modelo, es posible crear y escribir archivos propios de OCL, de modo que se puedan definir reglas específicas para un modelo creado; reglas (Rules) que son evaluadas en la actualidad sintácticamente. Esta ultima característica, permite al usuario aprender a enriquecer el modelo con un soporte formal.

4 PAMPA implementado como un DSL

Los DSL (Domain Specific Languages) son lenguajes de tamaño pequeño, muy centrados en resolver algunos problemas claramente identificables en el campo de un ámbito de aplicación concreto. Esta nueva tecnología permite a los diseñadores de software construir modelos gráficos adaptables a un determinado dominio de aplicación y generar código fuente usando diagramas como notación. En este contexto, las DSL Tools de Microsoft permiten crear DSLs aprovechando los beneficios del entorno de desarrollo *Visual Studio 2005* que simplifican y reducen el tiempo requerido para tal tarea, al mismo tiempo que favorecen el entendimiento del DSL en particular, obviando detalles de implementación que podrían dificultar el desarrollo del mismo. En particular esta tecnología, que se puede considerar como "herramientas para crear herramientas", facilita la tarea de definir un DSL como así también reducir el esfuerzo y los conocimientos necesarios para crear los editores gráficos y compiladores.

En la creación de un DSL se deben considerar 3 partes principales:

1. Definición del ***Modelo de Dominio***: incluye concretar las clases pertenecientes al dominio de aplicación y sus relaciones (DomainModel.dsldm).
2. Definición de la ***notación***: se definen las formas gráficas ("Shapes") que representan a los elementos del dominio definido, como por ejemplo: rectángulos, círculos, rombos, etc. También se especifica éstos llevan texto y en qué
formato (Designer.dsldd).
3. ***Mapear*** los elementos del ***Modelo de Dominio*** con las formas definidas en la ***notación*** (Designer.dsldd).

4.1 Migrando PAMPA hacia los beneficios de DSL Tools

A continuación detallaremos los componentes de PAMPA aplicando DSL Tools como nueva forma de implementación:

- **Metamodelo UML:** Cada DSL tiene como base un modelo de dominio donde se especifican los elementos del lenguaje, así como los atributos y relaciones de estos elementos. De esta manera, para implementar PAMPA como un DSL se debe generar un modelo de dominio que represente el metamodelo UML.
- **Administrador de Proyectos:** El DSL se encarga de la persistencia de los diagramas generados con el mismo, sin necesidad de codificación extra.
- **Editor Gráfico:** El Editor Gráfico es generado automáticamente por las DSL Tools a partir de la especificación de las formas (shape) que representan cada elemento del modelo de dominio del DSL. Además las DSL Tools ofrecen mecanismos de personalización gráfica para los elementos que dan como resultado diagramas verdaderamente ricos.
- **Editor de Expresiones OCL y Evaluador OCL:** Las DSL Tools de Microsoft aportan un framework para validar las reglas de buena formación. Las restricciones se pueden implementar como: *Métodos de Validación*, *Restricciones duras* o *Propiedades computadas*.

5 Aspecto educativo de la herramienta

Dotar a los estudiantes de los mecanismos y conocimientos necesarios para adoptar la metodología MDA, los ayudará a resolver sistemas de gran complejidad, diseñando la solución en el nivel adecuado de abstracción. Por lo tanto, el foco del desarrollo deja de estar en la escritura de código para una plataforma específica y pasa a centrarse en la creación de una solución de diseño efectiva a nivel de modelado.

Por otro lado, la combinación de notaciones gráficas intuitivas y los formalismos matemáticamente precisos provee claros beneficios cuando es tenida en cuenta para la definición del contenido de cursos sobre lenguajes de modelado como UML. El entendimiento formal de éste lenguaje facilita el uso apropiado de las herramientas CASE que lo soportan, dado que están basados en la definición formal de su sintaxis y semántica. El estudiante encuentra más fácil trabajar con la herramienta si conoce UML desde su punto de vista formal. Sin este conocimiento, es posible hacer uso de la herramienta pero no se conoce el "por qué" detrás de su funcionamiento.

PAMPA, al combinar el modelado gráfico de UML con la definición formal provista por OCL, surge como un recurso valioso para el aprendizaje y entendimiento profundo de los conceptos del modelado UML. Su utilización como recurso didáctico en seminarios introductorios de aplicación de lenguaje de modelado en el proceso de desarrollo de software tiene

como objetivo enseñar tanto los aspectos formales como gráficos de UML, abarcando no solo sus conceptos básicos (notación y uso de diagramas) sino además temas más avanzados como conceptos y definición del meta modelado, refinamiento de modelos, "customización" de restricciones formales a los modelos y validaciones formales de los mismos.

6 Lineas de trabajo futuro y conclusiones

Las funcionalidades anteriormente descriptas en la sección 3.1 sólo cubren los aspectos que consideramos básicos para la propuesta MDA, la edición de modelos con UML y su enriquecimiento con técnicas formales. Por este motivo consideramos que la herramienta debe ser ampliada con las siguientes funcionalidades a desarrollar en el futuro:

- *Migración del metamodelo*: traspaso del metamodelo de especificación UML 1.5 a UML 2.0.
- *Integración con Microsoft Visual Studio*: Al ser una herramienta implementada en la plataforma .NET, se considera de gran utilidad la futura integración al entorno de programación de Visual Studio.
- *Transformación de Modelos*: Transformar modelos PIM en uno o más modelos en distintas plataformas específicas (.Net, EJB, etc.).
- *Generación de Código*: posibilitando interactuar con diferentes generadores de código como C#, J#, VisualBasic.Net, y otros, a fin de obtener PSMs en esas plataformas.
- *Serialización de Modelos en XMI* (Xml Metadata Interface) [14]: facilitando el intercambio de modelos con otras herramientas de modelado.
- *Refinamiento de Modelos*: Este componente inspeccionará el modelo para descubrir y revelar casos de refinamientos ocultos.
- *Verificación Formal de pasos de Refinamiento*: permitiendo probar la existencia de una relación bien formada entre elementos del modelo.
- *Traducción de Invariantes y Aserciones de prueba*: Tomar expresiones OCL adjuntas a un artefacto abstracto y traducirlas a un vocabulario concreto, siguiendo los pasos de refinamiento.

6.1 Implementación de las funcionalidades futuras con DSL Tools:

Además de facilitar la implementación de las funciones que componen PAMPA actualmente, las DSL Tools de Microsoft permiten desarrollar fácilmente funcionalidades que estaban planeadas para el futuro de PAMPA:

1) *Migración del metamodelo UML 1.5 a UML 2.0*. Para ello se creará el ***modelo de dominio*** correspondiente al metamodelo de UML 2.0
2) *Integración con Microsoft Visual Studio.* Los DSL creados con las DSL Tools de Microsoft se integran fácilmente en el entorno de desarrollo de Visual Studio 2005.
3) *Transformación de Modelos.* Para las transformaciones desde un *modelo PSM* a un *modelo CODE* la implementación se basará en el desarrollo de ***templates*** para la generación de código.
4) *Generación de Código* (C#, J#, VisualBasic.Net, etc.). A fin de obtener PSMs en estas plataformas.: del mismo modo que en el punto 3), esto se logra escribiendo *templates* en cada uno de los lenguajes de programación deseados.
5) *Serialización de Modelos en XMI.* Con la definición de *templates* también se podrá generar código XMI a partir de los elementos del dominio instanciados en un diagrama UML.

6.2 Conclusiones finales

La comprensión y realización de modelos para la construcción de sistemas de software precisos y consistentes resulta clave en la visión MDA propuesta por OMG en el marco de la actual filosofía de desarrollo de software dirigido por modelos. Surge así la necesidad de aprender

y profundizar no sólo en los elementos y usos de un lenguaje de notación gráfica para modelar software -como el estándar UML- sino además, en la definición formal de su sintaxis y semántica, expresada en un lenguaje de especificación formal -como OCL- ampliamente aceptado. Para posibilitar tal combinación entre notaciones gráficas intuitivas y especificaciones formales, en este artículo hemos presentado la herramienta PAMPA como una forma clara de encarar estas problemáticas desde un punto de vista pedagógico dentro del contexto de MDD. La edición visual de diagramas UML y su validación formal enriquecida a través de expresiones OCL -que permiten aprender el uso adecuado del lenguaje-, constituyen los puntos claves de la funcionalidad de esta herramienta.

El manejo maduro de las notaciones gráficas y el entendimiento más profundo de los conceptos de formalismos aplicados a modelos, posibilitados por PAMPA, pretenden dar soporte a la producción de Modelos Independientes de cualquier Plataforma: precisos, consistentes y reusables, dentro del ciclo de vida del proceso MDA. Todo esto conduce a potencializar el uso de diversos elementos que fundamentan a MDA, tales como: modelos expresivos en UML, transformación de modelos PIM a PSM, especificación en lenguajes formales y generación automática de código.

Referencias bibliográficas

[1] Workshop on Model Driven Development (WMDD 2004) - June 15, 2004 - at ECOOP 2004, Oslo, Norway (June 14–18, 2004) - http://heim.ifi.uio.no/~janoa/wmdd2004

[2] UML 2.0 – The Unified Modeling Language Superstructure (formal/July 2005) and Infrastructure (formal/July 2005) Specification version 2.0 – OMG Final Adopted Specification. http://www.omg.org/technology/documents/formal/uml.htm

[3] OCL 2.0 – OMG Final Adopted Specification – June 2005 - http://www.omg.org/technology/documents/formal/uml.htm

[4] PAMPA - http://frlp.utn.edu.ar/pampa - L. Cuaderno, E. Di Lorenzo, A. Gaig, D. García, R. Giandini, L. Nahuel, L. Ocaranza, M. Pinasco, C. Pons, F. Salvatierra

[5] MDA Guide, v1.0.1, omg/03-06-01, June 2003. http://www.omg.org

[6] Object Managment Gruop - http://www.omg.org

[7] OMG, MOF Guide, v1.3 specification, formal/ April 2000 - http://www.omg.org/technology/cwm

[8] 1.1 Specification. http://www.jeckle.de/uml_spec.htm#umlv1.1

[9] MDR (Meta-Data Repository) http://mdr.netbeans.org/

[10] The Z notation. http://vl.zuser.org

[11] First Steps in Latex . George Gratzer. Birkhäuser,1999

[12] The C# Programming Language. Anders Hejlsberg, Scott Wiltamuth, Peter Golde. Addison-Wesley Professional, 2003.

[13] DB4Objects. http://www.db4o.com

[14] XMI - http://www.omg.org/technology/documents/formal/xmi.htm

[15] http://msdn2.microsoft.com/en-us/vstudio/aa718368.aspx